ASYNCHRONOUS REASONING: TRAINING-FREE INTERACTIVE THINKING LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Many state-of-the-art LLMs are trained to think before giving their answer. Reasoning can greatly improve language model capabilities, but it also makes them less interactive: given a new input, a model must stop thinking before it can respond. Real-world use cases such as voice-based or embodied assistants require an LLM agent to respond and adapt to additional information in real time, which is incompatible with sequential interactions. In contrast, humans can listen, think, and act asynchronously: we begin thinking about the problem while reading it and continue thinking while formulating the answer. In this work, we augment LLMs capable of reasoning to operate in a similar way without additional training. Our method uses the properties of positional embeddings to enable LLMs built for sequential generation to simultaneously think, listen, and write outputs. We evaluate our approach on math, commonsense, and safety reasoning: it allows models to generate accurate thinking-augmented answers while reducing time to first non-thinking token from minutes to $\leq$ 5s and the overall delays by up to $12\times$.

1 INTRODUCTION

Modern language models solve complex tasks using inference-time computation mechanisms (Snell et al., 2025; Suzgun et al., 2023; Beeching et al., 2024), such as chain-of-thought reasoning (Wei et al., 2022; Kojima et al., 2022; Yao et al., 2023a; Lightman et al., 2024) and agentic tool use (Schick et al., 2023; Yao et al., 2023b; Gao et al., 2023). Recent models, both proprietary (OpenAI et al., 2024; Google DeepMind, 2025; Anthropic, 2025) and open (Guo et al., 2025a; Qwen Team, 2025; Kimi Team et al., 2025), are explicitly trained for reasoning and agentic capabilities. As we trust these models with harder problems (ARC Prize Foundation, 2024; HLE Contributors, 2025), their ability to "think" becomes ever more important. The current dominant strategy for large language model (LLM) reasoning is the *read-think-answer* cycle: the model encodes a given problem, generates chain-of-thought reasoning, possibly calls tools, and then formulates the final answer (OpenAI et al., 2024; Guo et al., 2025a; Kimi Team et al., 2025). This paradigm naturally fits the sequential view of LLMs as next token prediction models. However, it also means that the LLM must follow a rigid turn structure that can limit its flexibility. The "thinking" phase can take minutes of real time, during which the agent does not get new information or output its current results.

By contrast, people have an innate ability to think asynchronously (Landi et al., 2013; Lyu et al., 2023; Trasmundi & Toro, 2023; Akiba, 2025). We can begin solving a problem even before hearing its entire statement. Moreover, we might start talking (or acting) while still completing our solution. Such "multitasking" is not always easy or efficient (Madore & Wagner, 2019), but it allows us to operate effectively in a dynamic environment (Corbetta et al., 2008).

Similarly, agents often need the ability to dynamically change their course of action. A voice assistant is expected to maintain a conversation in real time (OpenAI, 2024; Rubenstein et al., 2023; Zhang et al., 2023b; Chu et al., 2023; Xie & Wu, 2024; Fang et al., 2025; Défossez et al., 2024). The VLA model of an embodied agent needs to quickly adjust to new inputs (Ahn et al., 2022; Zitkovich et al., 2023; Driess et al., 2023). Fully text-based "deep research" agents benefit from interactive communication with the user (OpenAI, 2025). However, the standard read-think-answer cycle is non-interactive. During thinking, if an agent receives new inputs or must take action, it can either stop reasoning, discarding any incomplete thoughts, or wait until this phase is complete, sacrificing interactivity. As a result, many real-time LLM applications do not fully benefit from thinking.

In this work, we propose a technique that enables asynchronous LLM reasoning. Instead of retraining the model to satisfy each specific degree of interactivity, we propose an approach that leverages

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

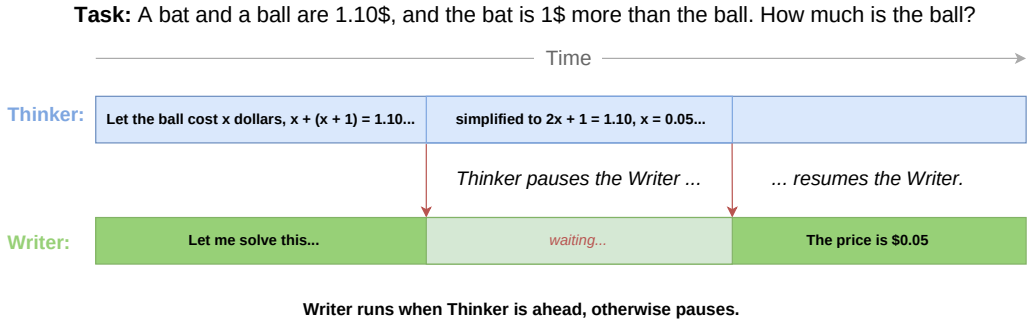

Figure 1: An intuitive explanation of asynchronous reasoning: the model generates its response concurrently with thinking. If the thinking stream needs additional time, it can pause the writing stream until the next reasoning step is ready. See animated `demo.gif` in the supplementary archive.

existing models, only changing their operation at inference time. Our approach uses three concurrent token streams: user inputs, private thoughts, and a public response, all of which can be updated in real time. We rely on the geometric properties of rotary positional embeddings to make the LLM perceive these streams as a single contiguous sequence *without additional training*. The model itself can decide whether it continues talking or pauses and thinks, depending on the current state of the three streams. The resulting method can be formulated as standard LLM inference with a modified attention cache, making it easy to integrate our approach into efficient LLM inference frameworks (Kwon et al., 2023; Zheng et al., 2024). Our main contributions can be summarized as follows:

- We propose AsyncReasoning, a zero-shot method that allows existing reasoning LLMs to think, write outputs, and encode extra inputs concurrently. Our approach relies on model-agnostic concurrent attention and zero-shot mode switching, making it easy to adapt to new models.
- We evaluate AsyncReasoning on multiple real-time benchmarks in mathematical and commonsense reasoning. Our experiments demonstrate that the proposed approach lets the LLM overlap thinking and answering, reducing time to first token by up to $80\times$ and total user-perceived delay by $12\times$ while retaining accuracy gains from reasoning.
- We demonstrate how AsyncReasoning can be used to improve model safety by thinking about risks in the background. This allows streaming real-time outputs on benign requests, while considering the safety implications in a private thinking stream that can pause potentially harmful outputs.
- Finally, we evaluate on tasks where the user specifies additional information after the initial prompt, such as clarifications or error correction. We found that modern LLMs with AsyncReasoning can incorporate this information on the fly without interrupting reasoning.
- We provide a reference implementation of AsyncReasoning, including GPU kernels for concurrent attention and an asynchronous voice assistant, in the anonymized supplementary materials.

2 Related Work

2.1 Real-time LLM Applications

LLM agents are deployed in a wide range of applications that require varying degrees of interactivity. Even text-based AI assistants often interrupted by users to provide corrections (McCain et al., 2026). Here, we briefly review several LLM applications that require quick or interactive responses.

Voice assistants. Recent works (Rubenstein et al., 2023; Zhang et al., 2023b; OpenAI, 2024; gem, 2024; Anthropic, 2025) use LLM agents as interactive voice assistants that talk to users in real time, often through edge devices, or participate in a group conversation (Flamino et al., 2025; Houde et al., 2025). Compared to text-based agents, voice assistants need faster reaction time, with the user often adding information while the agent is thinking. There are two main strategies for building voice assistants: modular and end-to-end. The first strategy feeds the output of automated speech recognition (Davis et al., 1952; Povey et al., 2011; Schneider et al., 2019; Radford et al., 2023) into a text-based LLM, then sends its response into a text-to-speech (TTS) system (Umeda et al., 1968; Zen et al., 2009; van den Oord et al., 2016; Wang et al., 2017; Shen et al., 2018; Prenger et al., 2019; Kong et al., 2020; Betker, 2023). The pipeline overlaps the LLM generation with TTS to stream audio in real time. The second, more recent, strategy is using language models that are trained to process and generate audio natively (Chu et al., 2023; Défossez et al., 2024; Xie & Wu, 2024; Fang et al., 2025). However, due to the constraints on response time, many speech LMs are not trained for long-form reasoning, and thinking models often cannot generate speech. For example, in the recent Qwen3-Omni family (Xu et al., 2025a), the 30B-A3B-Instruct can speak, but does not generate reasoning, while the 30B-A3B-Thinking has no speech *output*.

Robotic & virtual agents. Another type of LLM applications that requires interactivity is agents in real-time environments. Agents controlling robotic systems use Embodied Language Models (Driess et al., 2023; Ahn et al., 2022; Mon-Williams et al., 2025; Wang et al., 2024a; Jiang et al., 2023) for action planning or Vision-Language-Action (Zitkovich et al., 2023; Kim et al., 2025; Sapkota et al., 2025) models to control the system directly. Similar agents have been proposed for videogames (Wang et al., 2023), or system operations and mobile devices (Cao et al., 2024; Wu et al., 2024; Zhang et al., 2023a; Singer et al., 2025). Similarly, embodied agents need to react to stimuli from the environment.

Reasoning and Safety. Multiple works have studied the interactions between reasoning and the safety of responses (Korbak et al., 2025; Baker et al., 2025). Thinking can both mitigate safety risks and create new ones (Lu et al., 2025; Chua et al., 2025; Yang, 2025). However, when specifically prompted to reason about the safety, language models can detect and prevent jailbreak attacks (Zhou et al., 2025; Lou et al., 2025; Wu et al., 2025b; Zhang et al., 2025c). We review these works in Appendix K. We show that LLMs can reason about the safety of the user's request in the background, mitigating jailbreaks without response delays, which is inconvenient for interactive usage.

2.2 Efficient LLM Reasoning

There is a wide range of tasks that require LLMs to reason in real time. However, most thinking models (OpenAI et al., 2024; Guo et al., 2025a; Yang et al., 2025a) follow a non-interactive read-think-answer cycle. When receiving new information mid-thought, such LLMs can either interrupt their reasoning to react, sacrificing any incomplete thought tokens, or continue reasoning non-interactively.

Recently, there has been a large influx of techniques for efficient reasoning (Sui et al., 2025) through more concise chain-of-thought (Xu et al., 2025b; Aytes et al., 2025; Li et al., 2025), adaptive reasoning effort (Liang et al., 2025b; Zhang et al., 2025b; Zheng et al., 2025a) or early stopping (Pu et al., 2025; Sun et al., 2025; Laaouach, 2025). Another line of work explores reasoning in parallel (Ning et al., 2024; Jin et al., 2025; Rodionov et al., 2025; Yu, 2025; Hsu et al., 2025; Zheng et al., 2025b), or parallel tool calling (Gim et al., 2024; Kim et al., 2024). In this work, we focus on an orthogonal direction: we let the LLM formulate its response concurrently with its reasoning to reduce delays.

Reducing reasoning-induced delays. Several recent studies propose techniques that reduce the reasoning delays with partial read/think overlapping (Tong et al., 2025) or specialized two-model architectures with fast interactive and slow reasoning modules (Wu et al., 2025a). A concurrent work (Liang et al., 2025a) introduced a method that fine-tunes reasoning LLMs to solve their task with *interleaved* thinking and talking sub-blocks, making these models more interactive. Note that all these techniques require specialized fine-tuning or training from scratch, which complicates their adoption. Instead, we design an asynchronous reasoning method that does not require training.

3 AsyncReasoning

To convert a reasoning LLM into an asynchronous thinker, we need to formulate concurrent reasoning in a way that is compatible with the prompting structure the model was trained with. In Section 3.1, we describe how to dynamically rearrange the model's context so that it views two asynchronous streams as one sequence. In Section 3.2, we discuss mode switching: allowing the LLM to alternate between concurrent writing and waiting for thoughts to finish. Finally, we discuss efficient parallel token processing and other implementation details in Section 3.3.

3.1 Dual Thinker & Writer Views

The core idea behind our approach is that transformer LLMs are designed to manipulate sets (Vaswani et al., 2017; Lee et al., 2019), and the only thing that makes them aware of *sequence order* is positional encoding (Shaw et al., 2018; Press et al., 2022; Su et al., 2024). To change the order of elements in the prefix , we do not need to physically rearrange tokens in memory. It is sufficient to change positional relations between tokens, since the rest of the transformer architecture is already position-invariant.

At each inference step, AsyncReasoning manipulates positional representations to rearrange past tokens into a different order for thinking and for writing the response. Public response tokens "see" the (partial) private thoughts as if they were generated in a standard read-think-answer cycle. In turn, tokens within the `<think>` block "see" the response as if it was generated during the previous conversation turn. We illustrate this dual view in Figure 2 and provide prompts in Appendix A. This approach allows both "streams" (thinking and response) to immediately attend to each other's tokens as they are generated. The response tokens can "see" and use the latest thoughts without synchronization delays. The thinking stream "sees" the response and can pause the model's output if further thinking is necessary. This also allows our implementation to encode each token once and rearrange them using the geometry of relative positional embeddings (see Section 3.3).

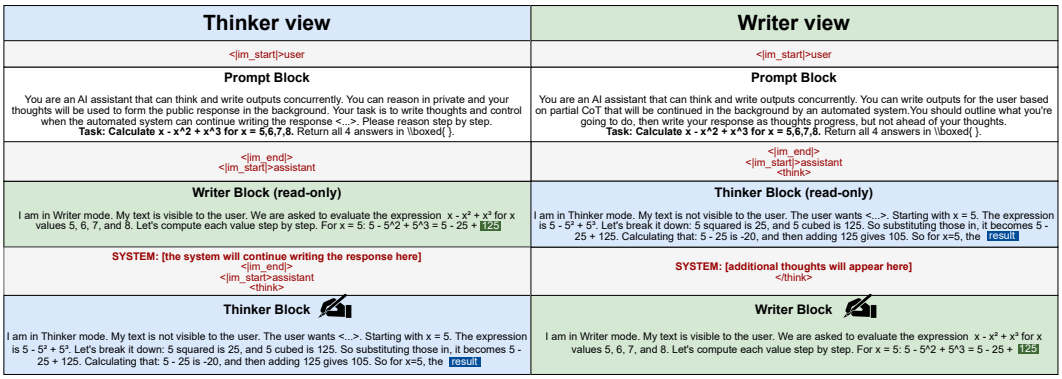

Figure 2: A dual thinker / writer view of the same reasoning task. The two views reuse the same KV cache and generate tokens in parallel. Both thinker and writer views match the model's chat template.

3.2 Mode Switching

Another important challenge of asynchronous thinking is deciding when to synchronize. Depending on the task at hand, the thinking stream may encounter a subtask that needs more "thinking time" to complete. If this is the case, the agent should briefly pause writing the response[1] and wait for the chain of thought to progress. AsyncReasoning lets the model itself control synchronization.

To achieve this, we periodically ask the model if its private thoughts are still ahead of the public response, or if it should pause and think more. Specifically, we insert a special prompt[2] into the thinking stream and compare the probabilities of "yes" vs. "no" as the next token. If the "yes" token is more likely, we keep thinking asynchronously. Otherwise, we pause the response stream until the model outputs "yes" again.

In our current implementation, we insert this question at the end of every paragraph or after every $T{=}20$ thinking tokens, whichever comes first. Crucially, once the model gives its "yes" or "no" response, we remove this prompt from its view (by hiding the corresponding key-value entries) so that it does not interfere with the model's chain-of-thought. See Appendix B for more details.

We compare different mode-switching prompts in Section 4.1. Overall, we found that existing thinking LLMs can already control asynchronous reasoning, though sometimes they do make mistakes (we elaborate on that in Appendix C). It is possible to design more advanced mechanisms, such as allowing the LLM to reason about mode switching in parallel or introducing a classifier "head" to decide when to pause responding. However, we opt to keep AsyncReasoning simple and training-free, deferring further study of mode switching to future work.

3.3 Implementation Details

In summary, AsyncReasoning arranges the thinking and response tokens in a different order depending on the generation phase, processes both streams in parallel, and periodically prompts the model to decide if it should pause and think. Thus, our algorithm alternates between two modes: either thinking and writing concurrently, or simply thinking while the writing is paused. When only one stream is active, AsyncReasoning is equivalent to standard sequential LLM inference with a combined KV cache. We focus the rest of this section on handling *concurrent* token streams.

We implement concurrent thinking and writing by creating a custom key-value cache and adjusting the positional embeddings to account for the dual views from Figure 2. The main purpose of this algorithm is to avoid redundant computation and the key-value cache bloat. Instead of encoding tokens twice for both views, we process each token exactly once and keep one KV cache entry that is "viewed" from different relative positions. This optimization is inspired by a similar rotation trick proposed in Hogwild! Inference (Rodionov et al., 2025), extended to support mode switching.

Key-Value Cache Structure. To implement different positional views, we split the model's KV cache into three contiguous "blocks" (tensors): the inputs, the thinking stream, and the output stream. As new tokens are generated or added by the user, we store them in the corresponding cache block

[1] For voice assistants, it may be better to communicate "Hmm, let me think about it...", but we do not do that.

[2] `"...\n\nWait, are my current thoughts enough to write the next paragraph or formula? (yes/no): "`

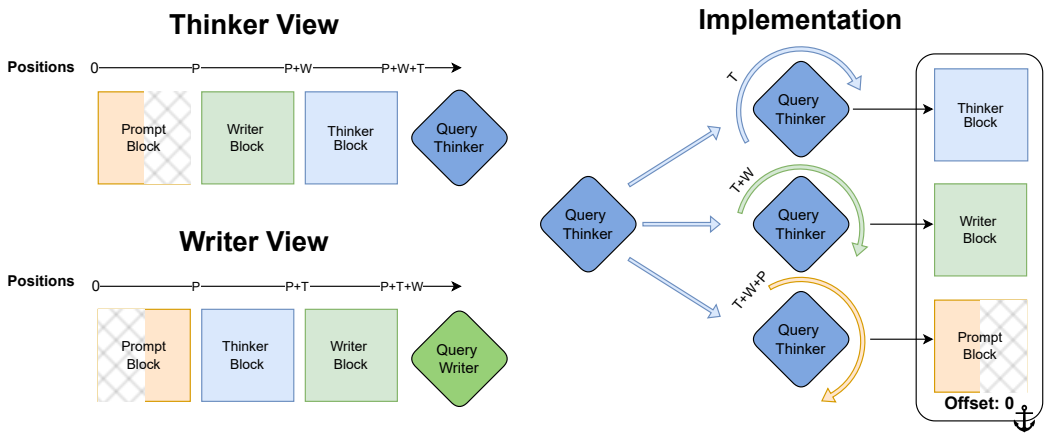

Figure 3: Concurrent thinking and writing implemented as batched inference. Newly added tokens attend to cache blocks with additional query rotations. Checkered areas represent masked tokens.

using positional representations relative to the block start[3]. During the self-attention forward pass, we concatenate the dot products between the query and all cache blocks. We transform the query differently for each block to simulate the difference in positions. This way, the same set of blocks can be combined for both thinking and writing views from Figure 2 without extra memory use.

Manipulating Positional Information. Almost all modern LLMs use some form of relative positional encoding (Shaw et al., 2018; Su et al., 2024; Press et al., 2022). The most popular variant is rotary positional embeddings (RoPE, Su et al., 2024), which rotate query and key vectors by an angle proportional to their index in the sequence. Because attention depends only on the difference between query and key positions, rotating the keys by $+\alpha$ is equivalent to rotating the queries by $-\alpha$.

We take advantage of this property to avoid rotating the entire KV cache on each inference step. Instead, we keep track of the starting positions for each block and rotate the attention queries. Suppose there are three contiguous KV blocks: **P**rompting with P tokens, **T**hinking with T tokens, and **W**riting with W tokens. When viewed contiguously (PTW), the difference between the most recent writer token and the thinker block is $T{+}W{-}1$ tokens. Thus, when running the forward pass for the *writer*, we rotate its query by the RoPE angle corresponding to position $T{+}W{-}1$ when looking at reasoning tokens and by $W{-}1$ when looking at writer's own tokens. In contrast, the *thinker* attends to itself at $T{-}1$ and to writer at $W{+}T{-}1$. The same principle applies to all query-key pairs.

Formally, let $\rho(q, i)$ denote applying RoPE for vector q at position i. The writer attends to blocks P, T, W: $A{:=}\rho(q, i_q){\cdot}\left[\rho(K_P, i_k^P), \rho(K_T, i_k^T), \rho(K_W, i_k^W)\right]$, where $[\cdot]$ denotes concatenation, i_q is the query position, i_k^P, i_k^T, i_k^W are cache block positions from the writer's point of view (see Figure 3) and $K_{P,T,W}$ are the corresponding key vectors. Then, we can equivalently compute attention as:

$$A{:=}\left[\rho(q, i_q{-}i_k^P)K_P, \rho(q, i_q{-}i_k^T)K_T, \rho(q, i_q{-}i_k^W)K_W\right].$$

In turn, thinker attends to the same KV cache entries with different query rotations corresponding to how they are arranged in its own view (Figure 3). This reformulation allows us to compute $K_{P,T,W}$ once, store them in the KV cache, and only modify the attention queries for the currently processed tokens during each forward pass. Appendix D extends this technique to non-RoPE models.

Technical Considerations. In summary, our implementation consists of a custom KV cache and an attention kernel that uses the query rotation trick described above. In practice, we use more than 3 KV blocks: in addition to the prompt, thinking and response tokens, we also have short linker tokens between thinking and writing blocks. These linkers are implemented as separate KV blocks that are visible only in one of the views (thinker or writer). If a block is not visible in the current view, we give it a large positional index to make it ignored due to causal attention masking.

This implementation can efficiently parallelize thinking and writing the response for small batch sizes. However, it can be optimized further for large batches by only processing the non-masked query-key pairs that actually contribute to the attention output. In future work, we plan to explore implementing more general kernels for AsyncReasoning based on vLLM's Paged Attention (Kwon et al., 2023).

[3]For example, given a model with RoPE embeddings, its KV cache will always store the 5th response token "rotated" for position 5, regardless of how many thinking tokens precede it.

4 EXPERIMENTS

We organize our experiments as follows: in Section 4.1, we verify the design choices for each component of AsyncReasoning. In Section 4.2, we expand to additional benchmarks and models. Section 4.3 tests the security of AsyncReasoning against adversarial attacks. Finally, Section 4.4 is focused on setups where the user provides additional clarifications after the initial prompt.

4.1 INITIAL ANALYSIS

We first evaluate different components of AsyncReasoning in detail with Qwen3-32B (Yang et al., 2025a), a popular medium-sized reasoning LLM that can run on a single high-end GPU. We run both AsyncReasoning and baselines on one A100-SXM4 GPU in `bfloat16` precision and greedy sampling. We do not use general inference optimizations such as speculative decoding, as they are orthogonal to our work. We evaluate on MATH-500 (Hendrycks et al., 2021; Lightman et al., 2024), a popular mathematical benchmark with "medium-difficulty" tasks that benefit from reasoning.

We focus on two main metrics: **accuracy**, computed using an LLM-as-a-judge protocol following the original benchmark setup, and **real-time delay**, defined as the amount of time (in seconds) during which the user hears no sound because the LLM has not generated the response yet. To evaluate accuracy, we prompt the LLM to put its answer in `\boxed{...}` in the public response and check its equivalence to the reference answer using the standard LLM-as-a-judge protocol[4] for MATH-500 (Zheng et al., 2023). To measure real-time delay, we stream the assistant's response to a TTS engine (see Section 3.3) and measure the total "silence time": the time during which the TTS could not generate anything because the LLM is still solving the task. We provide a more detailed description of the TTS pipeline and a more fine-grained performance breakdown in Appendix E.

We compare the following configurations:

1. **Baseline (Non-thinking):** regular sequential generation with `<think>` mode disabled.
2. **Baseline (Thinking):** regular sequential generation with `<think>` mode enabled.
3. **Interleaved Thinking:** prompting the model to think and reply in short, interleaved steps, without asynchrony. This setup is similar to Plantain (Liang et al., 2025a), but without model fine-tuning.
4. **DEER:** training-free dynamic early exit, recommended parameters from Yang et al. (2025b).
5. **ThinkSwitcher:** using a router that chooses: thinking or non-thinking mode (Liang et al., 2025c). We train it on MATH (train subset) using the protocol for Qwen3-8B from the original paper.
6. **AsyncReasoning:** the main setup from Section 3.2. The model is periodically asked whether the current thoughts are ahead of writing. If the answer is positive, the thinker and the writer both generate the next tokens in parallel. If not, the writer pauses until the next "yes" answer.
7. **AsyncReasoning ($+\Delta$):** same as above, but we increase the logit for answering "yes" to whether the writer should continue by a fixed bias $+\Delta$.
8. **AsyncReasoning (invert-q):** same as above, but the question is inverted:`"...\n\nWait, should I pause writing the response and think longer? (yes/no): "`.

The results in Figure 4 (left) show that AsyncReasoning can reduce real-time delays while preserving most of the accuracy gains from reasoning, outperforming non-asynchronous interleaved thinking. However, the exact tradeoff between accuracy and delay depends on the mode switching criterion and the bias. Our default criterion offers nearly the same accuracy as fully synchronous thinking, but at a lower real-time delay. The $+\Delta$ logit bias can further reduce this delay at the cost of some accuracy

[4]We use the protocol from `https://github.com/openai/simple-evals` with `gpt-4-turbo` as a judge.

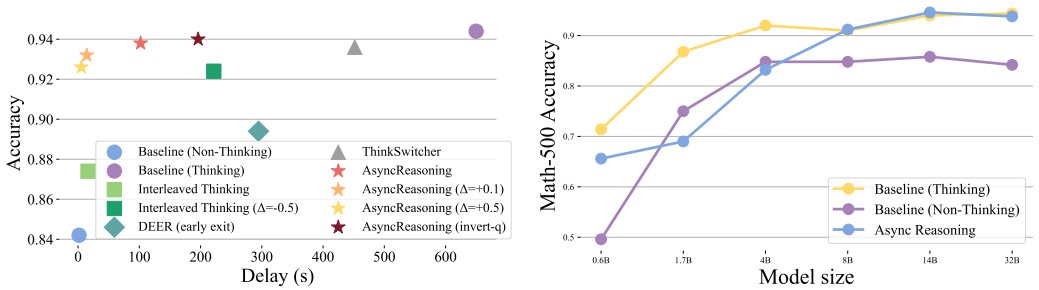

Figure 4: MATH-500 evaluation of AsyncReasoning and baselines on A100. Left: comparison of prompts and mode switching methods on Qwen3-32B. Right: performance across Qwen3 models.

loss, which comes from the writer answering too early. Flipping the question (where "yes" means pause) makes the model stop and think more often, suggesting that the model is biased to answer "yes". Unless stated otherwise, we use the main setup (no Δ or inversion) for the next sections. We include additional evaluations with other benchmarks, metrics (e.g., TTFT), stopping criteria and ablations on T in Appendix F as well as comparison against the trained Plantain model in Appendix J.

4.2 Additional Benchmarks

To evaluate AsyncReasoning generalization, we first considered established speech-language benchmarks (Yang et al., 2024; Wang et al., 2025; Wei et al., 2025; Yan et al., 2025), but modern models solve their harder tasks with near-perfect accuracy ($\geq 95\%$) *without thinking*. Following Shi et al. (2025), we use harder general benchmarks: MATH-500 (Hendrycks et al., 2021), MMLU-Pro (Wang et al., 2024c), GPQA-Diamond (Rein et al., 2023), AIME-2025 (AIME, 2025), ZebraLogic (Lin et al., 2025), and the voice-specific `spoken-mqa/multi_step_reasoning` (Wei et al., 2025). We report accuracy, total real-time delay, and TTFT (time-to-first-token); see Appendix G for details.

Table 1: Evaluation of AsyncReasoning on AIME-2025 (10 seeds) and ZebraLogic (grid) across different models using metrics from Section 4.2. ↑ (↓) arrows denote higher (lower) is better. Interactive baseline denotes no thinking for Qwen3 and low budget for GPT-OSS. †—Thinking 2507

	Model	Baseline (Thinking)			Baseline (Interactive)			Async Reasoning		
		Acc ↑	Delay ↓	TTFT ↓	Acc ↑	Delay ↓	TTFT ↓	Acc ↑	Delay ↓	TTFT ↓
AIME-2025	Qwen3-32B	0.53	1915.2	1914.4	0.20	1.70	1.13	0.49	485.93	5.41
	Qwen3-30B-A3B†	0.67	1814.1	1813.7	0.39	5.25	5.25	0.62	5.20	5.20
	Qwen3-235B-A22B†	0.68	5348.2	5330.5	0.53	20.16	9.73	0.68	500.94	9.06
	GPT-OSS-20B	0.62	523.53	523.53	0.36	120.09	120.09	0.59	147.96	24.15
	GPT-OSS-120B	0.77	326.52	326.52	0.51	91.40	91.40	0.66	166.75	24.63
ZebraLogic	Qwen3-32B	0.71	297.61	297.61	0.37	4.27	4.27	0.68	15.83	4.37
	Qwen3-30B-A3B†	0.96	1244.1	1244.1	0.27	4.16	4.16	0.93	3.72	3.72
	Qwen3-235B-A22B†	0.88	2111.7	2111.7	0.67	4.68	4.53	0.85	57.85	8.37
	GPT-OSS-20B	0.70	372.41	372.41	0.01	149.44	149.44	0.61	163.34	4.37
	GPT-OSS-120B	0.81	268.83	268.83	0.48	248.13	248.13	0.75	131.57	7.53

We use three model families: the original Qwen3 family from 0.6B to 32B (Yang et al., 2025a), Qwen3 (2507) models in 30B-A3B and 235B-A22B sizes, and GPT-OSS-20B/120B (OpenAI et al., 2025). To better showcase the setups, we run Qwen3 30B-A3B (2507) and GPT-OSS-120B on an H100 GPU, the 235B-A22B model on a B200 GPU, and the rest on A100. To fit the 235B model into a single GPU, we use NF4 expert-only quantization (Dettmers et al., 2023) and Qwen3 MoE fused kernels (woct0rdho, 2025). GPT-OSS models come with native ≈4-bit quantization and kernels. Appendix F measures the impact of quantization, GPU, and batch size.

Table 2: Qwen3 SpokenMQA multistep reasoning.

Size	Method	Acc ↑	Delay ↓	TTFT ↓
0.6B	Baseline (Thinking)	0.70	201.53	201.52
0.6B	Baseline (no Think)	0.57	0.72	0.67
0.6B	AsyncReasoning	0.66	2.56	1.37
4B	Baseline (Thinking)	0.81	227.16	227.15
4B	Baseline (no think)	0.80	0.80	0.70
4B	AsyncReasoning	0.81	0.84	0.80

Our results for MATH-500, MMLU-Pro and GPQA-Diamond are summarized in Figure 4 (right) and Figure 5 (both), other model-benchmark pairs are gathered in Tables 1 and 2. To save space, we

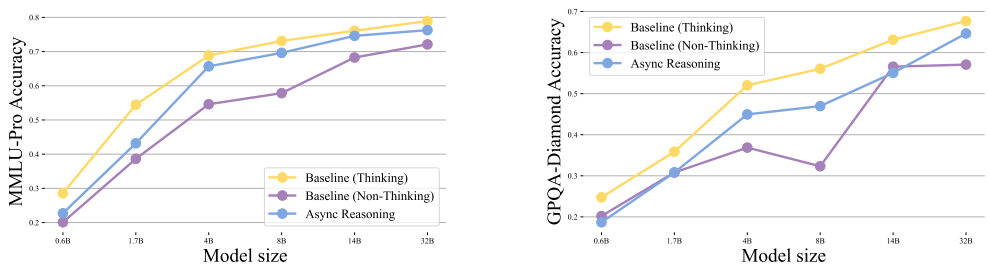

Figure 5: Evaluating AsyncReasoning and baselines across different Qwen3 model sizes on A100 GPU for two benchmarks. Left: MMLU-Pro. Right: GPQA-Diamond.

report larger models on AIME-2025 because smaller models score poorly regardless of reasoning type. Likewise, we focus on smaller models for SpokenMQA multistep reasoning since it contains relatively simple problems that do not need large model reasoning. That said, we report the remaining model-benchmark pairs in Appendix H (Tables 13—18) as well as aggregated summary in Appendix I.

Overall, we observe the same behavior as before: AsyncReasoning significantly reduces both time to first token and overall delays while providing more accurate answers than the non-thinking baseline. One notable exception is that smaller models (e.g. Qwen3-0.6B) lose more accuracy with asynchronous reasoning. On a closer examination, we found that many of the errors in smaller LLMs can be attributed to the writer giving the answer prematurely, suggesting that small models may struggle with mode switching. In future work, it would be interesting to revisit smaller models and see if their performance can be augmented with fine-tuning or training "mode-switching heads".

4.3 Asynchronous Reasoning about Safety

To evaluate the impact of asynchronous reasoning on safety, we conduct experiments on the HarmBench validation set (Mazeika et al., 2024). We use the first 200 samples focused on direct harm and jailbreaking attempts. We use LLM-as-a-judge (Zheng et al., 2023) evaluation with `gpt-4o`, where only actionable harmful instructions count as a successful attack.

We compare the Attack Success Rate (ASR) across the following setups using the Qwen3-32B model: (1) Baseline (non-thinking), (2) Baseline (thinking), (3) Baseline (thinking) with a safety prompt, (4) AsyncReasoning, and (5) AsyncReasoning with a safety prompt instructing the thinker to verify safety before responding (see Appendix A for full prompts). In addition to ASR and accuracy on MATH-500, we report time-to-first-token and total real-time delay to demonstrate that the safety benefits of AsyncReasoning come without sacrificing interactivity.

Table 3: Attack Success Rate on HarmBench and Accuracy on MATH-500 for Qwen3-32B.

Inference Setup	**ASR** $\downarrow$	**Accuracy** $\uparrow$
Baseline (non-thinking)	6.5%	0.84
Baseline (thinking)	12.5%	0.94
Baseline (thinking) + safety prompt	0.0%	0.93
AsyncReasoning (default)	10.0%	0.94
AsyncReasoning + safety prompt	**0.5%**	**0.93**

Table 3 summarizes our findings: consistent with "The Cost of Thinking" analysis (Yang, 2025), we observe that enabling reasoning in the baseline model actually *increases* vulnerability (ASR 6.5% $\rightarrow$ 12.5%). The model effectively "talks itself into" answering harmful queries by adopting a helpful persona or over-analyzing the technical aspects of the prompt. We provide an extended discussion of the complex interplay between reasoning and safety in Appendix K.

A safety-oriented prompt that asks the model to verify request safety before responding strongly mitigates these attacks both for synchronous reasoning (ASR 0%) and AsyncReasoning (ASR 0.5%). However, synchronous safety reasoning forces the user to wait for the entire chain-of-thought before any output appears, dramatically increasing real-time delays (Table 4). AsyncReasoning achieves comparable safety while reducing total delay from 45.3s to 35.0s and time-to-first-token from 32.3s to 22.2s, and also preserves accuracy on MATH-500 (0.93). We also observe similar trends on AdvBench (Zou et al., 2023), see Appendix M for details. This enables safety reasoning in streaming LLM APIs and other time-sensitive applications without the need for specialized fine-tuning. In Appendix L, we analyze model responses on successful attacks and identify the failure modes.

Table 4: Median latency on HarmBench for different setups, Qwen3-32B, A100.

Inference Setup	**TTFT (s)** $\downarrow$	**Total Delay (s)** $\downarrow$
Baseline (non-thinking)	0.1	6.1
Baseline (thinking)	41.3	61.1
Baseline (thinking) + safety prompt	32.3	45.3
AsyncReasoning + safety prompt	**22.2**	**35.0**

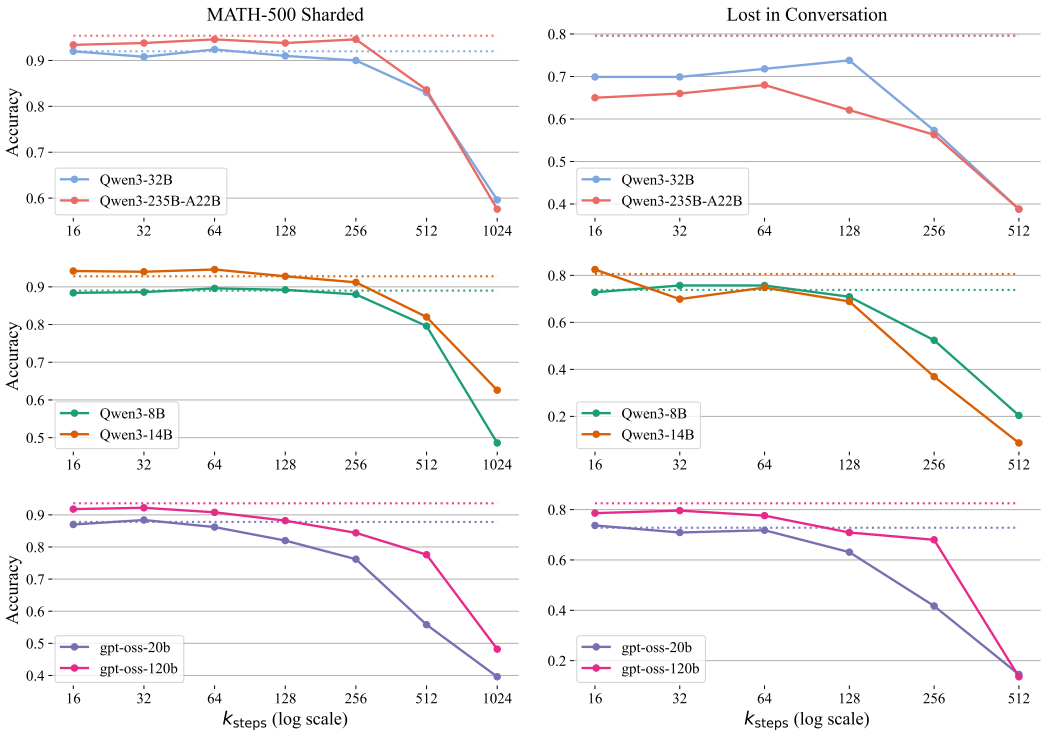

Figure 6: Evaluation with partial inputs on **sharded MATH-500** (left) and on the math subset of **`lost_in_conversation`** (right). The first shard is available immediately, subsequent shards are inserted every k_{steps}. Dotted lines denote accuracies without input sharding.

4.4 Asynchronous Reasoning with Additional Inputs

In real-time settings, additional inputs may arrive after decoding begins. We model these updates as *shards*, i.e., partial problem statements revealed over time. We insert shard i into the prompt, thinker, and writer cache blocks after $i \cdot k_{\text{steps}}$ decoding steps (only at paragraph boundaries `\n\n`), then continue generation without re-encoding (see Section 3.3); we vary k_{steps}. We derive a sharded dataset from MATH-500 by using `GPT-5` to rewrite each problem into two shards, where the second shard adds missing information or corrects an error. We verify the first shard alone is insufficient, while both shards are enough (Appendix N). We also evaluate on the math subset (103 samples) of `lost_in_conversation` (Laban et al., 2025) with more shards.

Across all models we evaluated, inserting each shard into all three blocks recovers accuracy on both datasets, approaching the upper bound where all shards are provided upfront for small k_{steps} (Figure 6). Accuracy drops as k_{steps} increases because more information arrives later; we do not handle shards arriving after generation completes, which may further reduce accuracy at large k_{steps}. The drop is larger on `lost_in_conversation`, consistent with its higher shard count (avg. ≈ 5.5). We analyze alternative strategies for additional inputs in Appendix O.

5 Discussion & Future Work

In this work, we formulated AsyncReasoning — a training-free method that allows reasoning LLMs to think and write concurrently. Our preliminary experiments suggest that the proposed approach can indeed overlap thinking and writing and reduce user delays while giving more accurate answers than the non-thinking models. This lets LLMs think longer and give better answers in time-sensitive tasks such as voice assistants, interactive agents, or safety-minded inference.

There are several directions for future research. First, it would be interesting to see if AsyncReasoning can be improved further with fine-tuning, and compare against baselines that require model training. This includes both fine-tuning/PEFT and ad-hoc "control heads" for mode switching. Another important direction is model safety: extending our initial setup to more comprehensive safety guardrails that use background reasoning to protect against a broader range of attacks. Additionally, we will work on integrating AsyncReasoning with SGLang (Zheng et al., 2024).

AI USE STATEMENT

In this work, we used generative AI tools for generating, cleaning, and reformatting a synthetic dataset, and a minor case of thematic data analysis. More specifically, we used LLMs to rewrite MATH-500 problems (Hendrycks et al., 2021) into pairs of partial or incorrect initial problems and correction messages, as described in Appendix N. We also used LLM-as-a-judge evaluation for benchmarks where this is part of our reported evaluation protocol. We also used generative AI tools for error analysis in our early experiments. We have not used generative AI tools for developing theoretical models or conceptual frameworks, implementing methods, proposing hypotheses, assisting with translation, feedback on research methodology, or interpreting results. Formulating mathematical claims or providing ingredients for these claims are not applicable to this work. Additionally, we used search-augmented generative AI tools to help find additional related works, and we used generative AI coding assistants to draft empty templates for certain tables and plots. We have reviewed all AI-assisted work: we checked the sharded MATH-500 samples through randomized manual inspection and by running evaluations to ensure that the LLM cannot produce the correct answer from incomplete first messages but a capable agent can produce the original correct answer from both shards. We also verified the table templates before filling them with our results. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies inference-time mechanisms for making reasoning LLMs more interactive. Such methods can improve responsiveness in applications such as voice assistants and embodied agents, but faster streaming also introduces risks. We discuss such potential consequences in dedicated Appendix R.

REPRODUCIBILITY STATEMENT

To facilitate reproducibility, we provide implementation code and experimental artifacts in the anonymized supplementary material. Appendices A–B contain prompts and mode switching details, Appendices E–J provide performance analysis, evaluation details, ablations, and complete results, Appendices N–O describe the sharded dataset and additional-input experiments, and Appendix P reports compute resources.

REFERENCES

Gemini live (voice mode). https://gemini.google/overview/gemini-live/, 2024. Accessed: 2025-12-01.

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as i can and not as i say: Grounding language in robotic affordances. In *arXiv preprint arXiv:2204.01691*, 2022.

AIME. Aime problems and solutions. https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions, 2025.

Daisuke Akiba. Ctrl + alt + inner speech: A verbal–cognitive scaffold (vcs) model of pathways to computational thinking. *Journal of Intelligence*, 13(12), 2025. ISSN 2079-3200. doi: 10.3390/jintelligence13120156. URL https://www.mdpi.com/2079-3200/13/12/156.

Anthropic. Claude 3.7 sonnet and claude code, 2025. URL https://www.anthropic.com/news/claude-3-7-sonnet. Accessed: 2025.04.02.

Anthropic. Using voice mode on claude mobile apps. https://support.claude.com/en/articles/11101966-using-voice-mode-on-claude-mobile-apps, 2025. Accessed: December 1, 2025.

ARC Prize Foundation. Openai's new o3 system scores breakthrough on arc-agi-pub, 2024. URL https://arcprize.org/blog/oai-o3-pub-breakthrough. Accessed: 2025.03.28.

Simon A. Aytes, Jinheon Baek, and Sung Ju Hwang. Sketch-of-thought: Efficient llm reasoning with adaptive cognitive-inspired sketching. *arXiv preprint arXiv:2503.05179*, mar 2025. doi: 10.48550/arXiv.2503.05179. URL https://arxiv.org/abs/2503.05179. Version 4 (last revised 24 Oct 2025).

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen Marcus McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. Llemma: An open language model for mathematics. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=4WnqRR915j.

Bowen Baker, Joost Huizinga, Leo Gao, Zehao Dou, Melody Y. Guan, Aleksander Madry, Wojciech Zaremba, Jakub W. Pachocki, and David Farhi. Monitoring reasoning models for misbehavior and the risks of promoting obfuscation. *ArXiv*, abs/2503.11926, 2025. URL https://api.semanticscholar.org/CorpusID:276937204.

Edward Beeching, Lewis Tunstall, and Sasha Rush. Scaling test-time compute with open models, 2024. URL https://huggingface.co/spaces/HuggingFaceH4/blogpost-scaling-test-time-compute.

James Betker. Better speech synthesis through scaling. *arXiv preprint arXiv:2305.07243*, 2023. Tortoise TTS: expressive multi-voice text-to-speech.

Charles Cao, Feiyi Wang, Lisa Lindley, and Zejiang Wang. Managing linux servers with llm-based ai agents: An empirical evaluation with gpt4. *Machine Learning with Applications*, 17: 100570, 2024. ISSN 2666-8270. doi: https://doi.org/10.1016/j.mlwa.2024.100570. URL https://www.sciencedirect.com/science/article/pii/S266682702400046X.

Edresson Casanova, Kelly Davis, Eren Gölge, Görkem Göknar, Iulian Gulea, Logan Hart, Aya Aljafari, Joshua Meyer, Reuben Morais, Samuel Olayemi, and Julian Weber. Xtts: a massively multilingual zero-shot text-to-speech model, 2024. URL https://arxiv.org/abs/2406.04904.

Wenhan Chang, Tianqing Zhu, Yu Zhao, Shuangyong Song, Ping Xiong, and Wanlei Zhou. Chain-of-lure: A universal jailbreak attack framework using unconstrained synthetic narratives, 2025. URL https://arxiv.org/abs/2505.17519.

Yunfei Chu, Jin Xu, Xiaohuan Zhou, Qian Yang, Shiliang Zhang, Zhijie Yan, Chang Zhou, and Jingren Zhou. Qwen-audio: Advancing universal audio understanding via unified large-scale audio-language models. arXiv preprint arXiv:2311.07919, 2023. URL https://arxiv.org/abs/2311.07919.

James Chua, Jan Betley, Mia Taylor, and Owain Evans. Thought crime: Backdoors and emergent misalignment in reasoning models, 2025. URL https://arxiv.org/abs/2506.13206.

Speech-Rule-Engine contributors. Speech-rule-engine: Generating speech descriptions for xml structures. GitHub repository, 2025. URL https://github.com/Speech-Rule-Engine/speech-rule-engine. accessed 2025-12-10.

Maurizio Corbetta, Gaurav Patel, and Gordon L Shulman. The reorienting system of the human brain: from environment to theory of mind. *Neuron*, 58(3):306–324, 2008.

Bobby Costin, Timor Averbuch, Dhruv Pai, Nathan Chen, and Ben Keigwin. Momoe: Memory optimized mixture of experts. *Tilde Research Blog*, 7 2025. URL https://www.tilderesearch.com/blog/momoe. Blog post.

K. H. Davis, R. Biddulph, and S. Balashek. Automatic recognition of spoken digits. *The Journal of the Acoustical Society of America*, 24(6):637–642, 11 1952. ISSN 0001-4966. doi: 10.1121/1.1906946. URL https://doi.org/10.1121/1.1906946.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 10088–10115. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1feb87871436031bdc0f2beaa62a049b-Paper-Conference.pdf.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. PaLM-e: An embodied multimodal language model. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 8469–8488. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/driess23a.html.

Alexandre Défossez, Laurent Mazaré, Manu Orsini, Amélie Royer, Patrick Pérez, Hervé Jégou, Edouard Grave, and Neil Zeghidour. Moshi: a speech-text foundation model for real-time dialogue, 2024. URL https://arxiv.org/abs/2410.00037.

ekwek1. Soprano: Instant, ultra-realistic text-to-speech. https://github.com/ekwek1/soprano, 2026. GitHub repository, accessed 2026-01-29.

Qingkai Fang, Yan Zhou, Shoutao Guo, Shaolei Zhang, and Yang Feng. LLaMA-omni 2: LLM-based real-time spoken chatbot with autoregressive streaming speech synthesis. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 18617–18629, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.912. URL https://aclanthology.org/2025.acl-long.912/.

James Flamino, Mohammed Shahid Modi, Boleslaw K. Szymanski, Brendan Cross, and Colton Mikolajczyk. Testing the limits of large language models in debating humans. *Scientific Reports*, 15:13852, 2025. doi: 10.1038/s41598-025-98378-1. URL https://doi.org/10.1038/s41598-025-98378-1.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10764–10799. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/gao23f.html.

In Gim, Seung seob Lee, and Lin Zhong. Asynchronous llm function calling, 2024. URL https://arxiv.org/abs/2412.07017.

Google DeepMind. Gemini 2.5: Our Newest Gemini Model with Thinking. https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/#gemini-2-5-thinking, 2025. Accessed: 2025-04-07.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu,

Tao Yun, Tian Pei, Tianyu Sun, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025a. doi: 10.1038/s41586-025-09422-z.

Shoutao Guo, Shaolei Zhang, Zhengrui Ma, and Yang Feng. Large language models are read/write policy-makers for simultaneous generation, 2025b. URL https://arxiv.org/abs/2501.00868.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021.

HLE Contributors. Humanity's last exam: A benchmark for frontier ai capabilities. *arXiv preprint arXiv:2501.14249*, 2025. doi: 10.48550/arXiv.2501.14249.

Stephanie Houde, Kristina Brimijoin, Michael Muller, Steven I. Ross, Dario Andres Silva Moran, Gabriel Enrique Gonzalez, Siya Kunde, Morgan A. Foreman, and Justin D. Weisz. Controlling ai agent participation in group conversations: A human-centered approach. In *Proceedings of the 30th International Conference on Intelligent User Interfaces*, IUI '25, pp. 390–408, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400713064. doi: 10.1145/3708359.3712089. URL https://doi.org/10.1145/3708359.3712089.

Chan-Jan Hsu, Davide Buffelli, Jamie McGowan, Feng-Ting Liao, Yi-Chang Chen, Sattar Vakili, and Da shan Shiu. Group think: Multiple concurrent reasoning agents collaborating at token level granularity, 2025. URL https://arxiv.org/abs/2505.11107.

Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. VIMA: Robot manipulation with multimodal prompts. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 14975–15022. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/jiang23b.html.

Tian Jin, Ellie Y. Cheng, Zack Ankner, Nikunj Saunshi, Blake M. Elias, Amir Yazdanbakhsh, Jonathan Ragan-Kelley, Suvinay Subramanian, and Michael Carbin. Learning to keep a promise: Scaling language model decoding parallelism with learned asynchronous decoding, 2025. URL https://arxiv.org/abs/2502.11517.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. Openvla: An open-source vision-language-action model. In Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard (eds.), *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2679–2713. PMLR, 06–09 Nov 2025. URL https://proceedings.mlr.press/v270/kim25c.html.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. An llm compiler for parallel function calling. In *Forty-first International Conference on Machine Learning*, 2024.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, Zhuofu Chen, Jialei Cui, Hao Ding, Mengnan Dong,

Angang Du, Chenzhuang Du, Dikang Du, Yulun Du, Yu Fan, Yichen Feng, Kelin Fu, Bofei Gao, Hongcheng Gao, Peizhong Gao, Tong Gao, Xinran Gu, Longyu Guan, Haiqing Guo, Jianhang Guo, Hao Hu, Xiaoru Hao, Tianhong He, Weiran He, Wenyang He, Chao Hong, Yangyang Hu, Zhenxing Hu, Weixiao Huang, Zhiqi Huang, Zihao Huang, Tao Jiang, Zhejun Jiang, Xinyi Jin, Yongsheng Kang, Guokun Lai, Cheng Li, Fang Li, Haoyang Li, Ming Li, Wentao Li, Yanhao Li, Yiwei Li, Zhaowei Li, Zheming Li, Hongzhan Lin, Xiaohan Lin, Zongyu Lin, Chengyin Liu, Chenyu Liu, Hongzhang Liu, Jingyuan Liu, Junqi Liu, Liang Liu, Shaowei Liu, T. Y. Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yibo Liu, Yiping Liu, Yue Liu, Zhengying Liu, Enzhe Lu, Lijun Lu, Shengling Ma, Xinyu Ma, Yingwei Ma, Shaoguang Mao, Jie Mei, Xin Men, Yibo Miao, Siyuan Pan, Yebo Peng, Ruoyu Qin, Bowen Qu, Zeyu Shang, Lidong Shi, Shengyuan Shi, Feifan Song, Jianlin Su, Zhengyuan Su, Xinjie Sun, Flood Sung, Heyi Tang, Jiawen Tao, Qifeng Teng, Chensi Wang, Dinglu Wang, Feng Wang, Haiming Wang, Jianzhou Wang, Jiaxing Wang, Jinhong Wang, Shengjie Wang, Shuyi Wang, Yao Wang, Yejie Wang, Yiqin Wang, Yuxin Wang, Yuzhi Wang, Zhaoji Wang, Zhengtao Wang, Zhexu Wang, Chu Wei, Qianqian Wei, Wenhao Wu, Xingzhe Wu, Yuxin Wu, Chenjun Xiao, Xiaotong Xie, Weimin Xiong, Boyu Xu, Jing Xu, Jinjing Xu, L. H. Xu, Lin Xu, Suting Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ziyao Xu, Junjie Yan, Yuzi Yan, Xiaofei Yang, Ying Yang, Zhen Yang, Zhilin Yang, Zonghan Yang, Haotian Yao, Xingcheng Yao, Wenjie Ye, Zhuorui Ye, Bohong Yin, Longhui Yu, Enming Yuan, Hongbang Yuan, Mengjie Yuan, Haobing Zhan, Dehao Zhang, Hao Zhang, Wanlu Zhang, Xiaobin Zhang, Yangkun Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Haotian Zhao, Yikai Zhao, Huabin Zheng, Shaojie Zheng, Jianren Zhou, Xinyu Zhou, Zaida Zhou, Zhen Zhu, Weiyu Zhuang, and Xinxing Zu. Kimi k2: Open agentic intelligence, 2025. URL https://arxiv.org/abs/2507.20534.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*, volume 35, pp. 22199–22213, 2022.

Jungil Kong, Jaehyeon Kim, and Jaekyoung Bae. Hifi-gan: generative adversarial networks for efficient and high fidelity speech synthesis. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Tomek Korbak, Mikita Balesni, Elizabeth Barnes, Yoshua Bengio, Joe Benton, Joseph Bloom, Mark Chen, Alan Cooney, Allan Dafoe, Anca Dragan, Scott Emmons, Owain Evans, David Farhi, Ryan Greenblatt, Dan Hendrycks, Marius Hobbhahn, Evan Hubinger, Geoffrey Irving, Erik Jenner, Daniel Kokotajlo, Victoria Krakovna, Shane Legg, David Lindner, David Luan, Aleksander Mądry, Julian Michael, Neel Nanda, Dave Orr, Jakub Pachocki, Ethan Perez, Mary Phuong, Fabien Roger, Joshua Saxe, Buck Shlegeris, Martín Soto, Eric Steinberger, Jasmine Wang, Wojciech Zaremba, Bowen Baker, Rohin Shah, and Vlad Mikulik. Chain of Thought Monitorability: A New and Fragile Opportunity for AI Safety. *arXiv*, 2025. URL https://arxiv.org/abs/2507.11473.

Martin Kuo, Jianyi Zhang, Aolin Ding, Qinsi Wang, Louis DiValentin, Yujia Bao, Wei Wei, Hai Li, and Yiran Chen. H-cot: Hijacking the chain-of-thought safety reasoning mechanism to jailbreak large reasoning models, including openai o1/o3, deepseek-r1, and gemini 2.0 flash thinking. *arXiv preprint*, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

Yassir Laaouach. HALT-cot: Model-agnostic early stopping for chain-of-thought reasoning via answer entropy. In *4th Muslims in ML Workshop co-located with ICML 2025*, 2025. URL https://openreview.net/forum?id=CX5c7C1CZa.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. Llms get lost in multi-turn conversation, 2025. URL https://arxiv.org/abs/2505.06120.

Eric Lam. lab-mic: Record audio directly within jupyter/ipython notebooks using browser microphone. GitHub repository. URL https://github.com/voidful/lab-mic. accessed 2025-12-10.

Nicole Landi, Stephen J. Frost, W. Einar Mencl, Rebecca Sandak, and Kenneth R. Pugh. Neurobiological bases of reading comprehension: Insights from neuroimaging studies of word-level and text-level processing in skilled and impaired readers. *Reading & writing quarterly : overcoming learning difficulties*, 29:145–167, 04 2013. doi: 10.1080/10573569.2013.758566.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 3744–3753, 2019.

Chengshu Li, Jacky Liang, Andy Zeng, Xinyun Chen, Karol Hausman, Dorsa Sadigh, Sergey Levine, Li Fei-Fei, Fei Xia, and Brian Ichter. Chain of code: Reasoning with a language model-augmented code emulator. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 28259–28277. PMLR, 2024. URL https://proceedings.mlr.press/v235/li24ar.html. arXiv preprint arXiv:2312.04474.

Gengyang Li, Yifeng Gao, Yuming Li, and Yunfang Wu. Thinkless: A training-free inference-efficient method for reducing reasoning redundancy. *arXiv preprint arXiv:2505.15684*, may 2025. doi: 10.48550/arXiv.2505.15684. URL https://arxiv.org/abs/2505.15684. Version 2 (last revised 23 May 2025).

Anthony Liang, Jonathan Berant, Adam Fisch, Abhimanyu Goyal, Kalpesh Krishna, and Jacob Eisenstein. Plantain: Plan-answer interleaved reasoning, 2025a. URL https://arxiv.org/abs/2512.03176.

Guosheng Liang, Longguang Zhong, Ziyi Yang, and Xiaojun Quan. Thinkswitcher: Dynamic switching between short and long chain-of-thought reasoning in large reasoning models, 2025b. arXiv preprint.

Guosheng Liang, Longguang Zhong, Ziyi Yang, and Xiaojun Quan. Thinkswitcher: When to think hard, when to think fast. *arXiv preprint arXiv:2505.14183*, 2025c.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Bill Yuchen Lin, Ronan Le Bras, Kyle Richardson, Ashish Sabharwal, Radha Poovendran, Peter Clark, and Yejin Choi. ZebraLogic: On the scaling limits of LLMs for logical reasoning. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 37889–37905. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/lin25i.html.

Xinyue Lou, You Li, Jinan Xu, Xiangyu Shi, Chi Chen, and Kaiyu Huang. Think in safety: Unveiling and mitigating safety alignment collapse in multimodal large reasoning model. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 5167–5186, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.261. URL https://aclanthology.org/2025.emnlp-main.261/.

Chengda Lu, Xiaoyu Fan, Yu Huang, Rongwu Xu, Jijie Li, and Wei Xu. Does chain-of-thought reasoning really reduce harmfulness from jailbreaking? In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6523–6546, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.339. URL https://aclanthology.org/2025.findings-acl.339/.

Bingjiang Lyu, William D. Marslen-Wilson, Yuxing Fang, and Lorraine K. Tyler. Finding structure during incremental speech comprehension. *eLife*, 12:e89311, 2023. doi: 10.7554/eLife.89311. URL https://elifesciences.org/articles/89311.

Kevin P. Madore and Anthony D. Wagner. Multicosts of multitasking. *Cerebrum : the Dana Forum on Brain Science*, 2019:cer–04–19, 2019. ISSN 1524-6205. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7075496/.

Mantas Mazeika, Long Phan, Xuwang Yin, Andy Zou, Zifan Wang, Norman Mu, Elham Sakhaee, Nathaniel Li, Steven Basart, Bo Li, David Forsyth, and Dan Hendrycks. HarmBench: A standardized evaluation framework for automated red teaming and robust refusal. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 35181–35224. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/mazeika24a.html.

Miles McCain, Thomas Millar, Saffron Huang, Jake Eaton, Kunal Handa, Michael Stern, Alex Tamkin, Matt Kearney, Esin Durmus, Judy Shen, Jerry Hong, Brian Calvert, Jun Shern Chan, Francesco Mosconi, David Saunders, Tyler Neylon, Gabriel Nicholas, Sarah Pollack, Jack Clark, and Deep Ganguli. Measuring ai agent autonomy in practice, 2026. URL https://anthropic.com/research/measuring-agent-autonomy.

Ruaridh Mon-Williams, Gen Li, Ran Long, Wenqian Du, Christopher G. Lucas, et al. Embodied large language models enable robots to complete complex tasks in unpredictable environments. *Nature Machine Intelligence*, 7:592–601, 2025. doi: 10.1038/s42256-025-01005-x.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*, 2025.

Xuefei Ning, Zinan Lin, Zixuan Zhou, Zifu Wang, Huazhong Yang, and Yu Wang. Skeleton-of-thought: Prompting LLMs for efficient parallel generation. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=mqVgBbNCm9.

OpenAI. Gpt-4o system card. Online technical report, 2024. URL https://openai.com/index/hello-gpt-4o. Voice-mode multimodal model supporting audio, text, and vision. Available at https://openai.com/index/hello-gpt-4o.

OpenAI. Chatgpt deep research: Support for user update and multitasking features. https://chat.openai.com, 2025. Accessed 7 December 2025. In late 2025, the Deep Research feature was updated to allow user to communicate with the agent while it performs research via the "Update" button.

OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, and Alex Beutel et al. Openai o1 system card, 2024. URL https://arxiv.org/abs/2412.16720.

OpenAI, :, Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K. Arora, Yu Bai, Bowen Baker, Haiming Bao, Boaz Barak, Ally Bennett, Tyler Bertao, Nivedita Brett, Eugene Brevdo, Greg Brockman, Sebastien Bubeck, Che Chang, Kai Chen, Mark Chen, Enoch Cheung, Aidan Clark, Dan Cook, Marat Dukhan, Casey Dvorak, Kevin Fives, Vlad Fomenko, Timur Garipov, Kristian Georgiev, Mia Glaese, Tarun Gogineni, Adam Goucher, Lukas Gross, Katia Gil Guzman, John Hallman, Jackie Hehir, Johannes Heidecke, Alec Helyar, Haitang Hu, Romain Huet, Jacob Huh, Saachi Jain, Zach Johnson, Chris Koch, Irina Kofman, Dominik Kundel, Jason Kwon, Volodymyr Kyrylov, Elaine Ya Le, Guillaume Leclerc, James Park Lennon, Scott Lessans, Mario Lezcano-Casado, Yuanzhi Li, Zhuohan Li, Ji Lin, Jordan Liss, Lily, Liu, Jiancheng Liu, Kevin Lu, Chris Lu, Zoran Martinovic, Lindsay McCallum, Josh McGrath, Scott McKinney, Aidan McLaughlin, Song Mei, Steve Mostovoy, Tong Mu, Gideon Myles, Alexander Neitz, Alex Nichol, Jakub Pachocki, Alex Paino, Dana Palmie, Ashley Pantuliano, Giambattista Parascandolo, Jongsoo Park, Leher Pathak, Carolina Paz, Ludovic Peran, Dmitry Pimenov, Michelle Pokrass, Elizabeth Proehl, Huida Qiu, Gaby Raila, Filippo Raso, Hongyu Ren, Kimmy Richardson, David Robinson, Bob Rotsted, Hadi Salman, Suvansh Sanjeev, Max Schwarzer, D. Sculley, Harshit Sikchi, Kendal Simon, Karan Singhal, Yang Song, Dane Stuckey, Zhiqing Sun, Philippe Tillet, Sam Toizer, Foivos Tsimpourlas, Nikhil Vyas, Eric Wallace, Xin Wang, Miles Wang, Olivia Watkins, Kevin Weil, Amy Wendling, Kevin Whinnery, Cedric Whitney, Hannah Wong, Lin Yang, Yu Yang, Michihiro Yasunaga, Kristen Ying, Wojciech Zaremba, Wenting

Zhan, Cyril Zhang, Brian Zhang, Eddie Zhang, and Shengjia Zhao. gpt-oss-120b & gpt-oss-20b model card, 2025. URL https://arxiv.org/abs/2508.10925.

Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely. The kaldi speech recognition toolkit. https://kaldi-asr.org, 2011. Open-source speech recognition toolkit.

Ryan Prenger, Rafael Valle, and Bryan Catanzaro. Waveglow: A flow-based generative network for speech synthesis. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3617–3621, 2019. doi: 10.1109/ICASSP.2019.8683143.

Ofir Press, Noah Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=R8sQPpGCv0.

Xiao Pu, Michael Saxon, Wenyue Hua, and William Yang Wang. Thoughtterminator: Benchmarking, calibrating, and mitigating overthinking in reasoning models, 2025. arXiv preprint.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023. URL https://arxiv.org/abs/2307.16789.

Qwen Team. Qwq-32b: Embracing the power of reinforcement learning, March 2025. URL https://qwenlm.github.io/blog/qwq-32b/.

Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine Mcleavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 28492–28518. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/radford23a.html.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark, 2023. URL https://arxiv.org/abs/2311.12022.

Gleb Rodionov, Roman Garipov, Alina Shutova, George Yakushev, Erik Schultheis, Vage Egiazarian, Anton Sinitsin, Denis Kuznedelev, and Dan Alistarh. Hogwild! inference: Parallel llm generation via concurrent attention, 2025. URL https://arxiv.org/abs/2504.06261.

Paul K. Rubenstein, Chulayuth Asawaroengchai, Duc Dung Nguyen, Ankur Bapna, Zalán Borsos, Félix de Chaumont Quitry, Peter Chen, Dalia El Badawy, Wei Han, Eugene Kharitonov, Hannah Muckenhirn, Dirk Padfield, James Qin, Danny Rozenberg, Tara Sainath, Johan Schalkwyk, Matt Sharifi, Michelle Tadmor Ramanovich, Marco Tagliasacchi, Alexandru Tudor, Mihajlo Velimirović, Damien Vincent, Jiahui Yu, Yongqiang Wang, Vicky Zayats, Neil Zeghidour, Yu Zhang, Zhishuai Zhang, Lukas Zilka, and Christian Frank. Audiopalm: A large language model that can speak and listen. arXiv preprint arXiv:2306.12925, 2023. URL https://arxiv.org/abs/2306.12925.

Ranjan Sapkota, Yang Cao, Konstantinos I. Roumeliotis, and Manoj Karkee. Vision-language-action models: Concepts, progress, applications and challenges, 2025. URL arxiv.org.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 68539–68551. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/d842425e4bf79ba039352da0f658a906-Paper-Conference.pdf.

Steffen Schneider, Alexei Baevski, Ronan Collobert, and Michael Auli. wav2vec: Unsupervised pre-training for speech recognition. In *Interspeech 2019*, pp. 3465–3469, 2019. doi: 10.21437/Interspeech.2019-1873.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In Marilyn Walker, Heng Ji, and Amanda Stent (eds.), *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pp. 464–468, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-2074. URL https://aclanthology.org/N18-2074/.

Jonathan Shen, Ruoming Pang, Ron J. Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, Rj Skerrv-Ryan, Rif A. Saurous, Yannis Agiomvrgiannakis, and Yonghui Wu. Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4779–4783, 2018. doi: 10.1109/ICASSP.2018.8461368.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yue Ting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *ArXiv*, abs/2303.17580, 2023. URL https://api.semanticscholar.org/CorpusID:257833781.

Yemin Shi, Yu Shu, Siwei Dong, Guangyi Liu, Jaward Sesay, Jingwen Li, and Zhiting Hu. Voila: Voice-language foundation models for real-time autonomous interaction and voice role-play, 2025. URL https://arxiv.org/abs/2505.02707.

Brian Singer, Keane Lucas, Lakshmi Adiga, Meghna Jain, Lujo Bauer, and Vyas Sekar. On the feasibility of using llms to execute multistage network attacks, 01 2025.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2023.127063. URL https://www.sciencedirect.com/science/article/pii/S0925231223011864.

Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Na Zou, Hanjie Chen, and Xia Hu. A survey on efficient reasoning for large language models. *arXiv preprint arXiv:2503.16419*, mar 2025. URL https://arxiv.org/abs/2503.16419. Version 4 (last updated August 21, 2025).

Renliang Sun, Wei Cheng, Dawei Li, Haifeng Chen, and Wei Wang. Stop when enough: Adaptive early-stopping for chain-of-thought reasoning, 2025. arXiv preprint.

Mirac Suzgun, Nathan Scales, Nathanael Scharli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13003–13051, Toronto, Canada, 2023. URL https://api.semanticscholar.org/CorpusID:252917648.

Junlong Tong, Yingqi Fan, Anhao Zhao, Yunpu Ma, and Xiaoyu Shen. Streamingthinker: Large language models can think while reading, 2025. URL https://arxiv.org/abs/2510.17238.

Sarah Bro Trasmundi and Juan Toro. Mind wandering in reading: An embodied approach. *Frontiers in Human Neuroscience*, 17, 2023. ISSN 1662-5161. doi: 10.3389/fnhum.2023.1061437. URL www.frontiersin.org.

Noriko Umeda, E Matsui, Torazo Suzuki, and Hiroshi Omura. Synthesis of fairy tales using an analog vocal tract. In *Proceedings of 6th International Congress on Acoustics*, pp. B159–162, 1968.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *Proceedings of the 9th ISCA Speech Synthesis Workshop*, 2016. URL https://arxiv.org/abs/1609.03499.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Bin Wang, Xunlong Zou, Geyu Lin, Shuo Sun, Zhuohan Liu, Wenyu Zhang, Zhengyuan Liu, AiTi Aw, and Nancy F Chen. Audiobench: A universal benchmark for audio large language models. *NAACL*, 2025.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research (TMLR)*, 2024a. doi: 10.48550/arXiv.2305.16291.

Ke Wang, Houxing Ren, Aojun Zhou, Zimu Lu, Sichun Luo, Weikang Shi, Renrui Zhang, Linqi Song, Mingjie Zhan, and Hongsheng Li. Mathcoder: Seamless code integration in llms for enhanced mathematical reasoning. In *International Conference on Learning Representations*, volume 2024, pp. 5009–5042, 2024b.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *arXiv preprint arXiv:2406.01574*, 2024c.

Yuxuan Wang, R. J. Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J. Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, Quoc Le, Yannis Agiomyrgiannakis, Rob Clark, and Rif A. Saurous. Tacotron: Towards end-to-end speech synthesis. In *INTERSPEECH*, pp. 4006–4010. ISCA, 2017. doi: 10.21437/Interspeech.2017-1452. URL https://arxiv.org/abs/1703.10135.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian (Shawn) Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with llms enables open-world multi-task agents. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 34153–34189. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/6b8dfb8c0c12e6fafc6c256cb08a5ca7-Paper-Conference.pdf.

Chengwei Wei, Bin Wang, Jung-jae Kim, and Nancy F. Chen. Towards spoken mathematical reasoning: Benchmarking speech-based models over multi-faceted math problems. *arXiv preprint arXiv:2505.15000*, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

woct0rdho. woct0rdho/transformers-qwen3-moe-fused: Fused qwen3 moe layer for faster training, compatible with transformers, lora, bnb 4-bit quant, unsloth, 2025. URL https://github.com/woct0rdho/transformers-qwen3-moe-fused/. GitHub repository, Apache-2.0 license.

Donghang Wu, Haoyang Zhang, Jun Chen, Xiangyu, Zhang, Hexin Liu, Eng Siong Chng, Fei Tian, Xuerui Yang, Xiangyu Zhang, Daxin Jiang, and Gang Yu. Mind-paced speaking: A dual-brain approach to real-time reasoning in spoken language models, 2025a. URL https://arxiv.org/abs/2510.09592.

Tong Wu, Chong Xiang, Jiachen T. Wang, G. Edward Suh, and Prateek Mittal. Effectively controlling reasoning models through thinking intervention, 2025b. URL https://arxiv.org/abs/2503.24370.

Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. Os-copilot: Towards generalist computer agents with self-improvement, 2024.

Zhifei Xie and Changqiao Wu. Mini-omni: Language models can hear, talk while thinking in streaming. arXiv preprint arXiv:2408.16725, 2024. URL https://arxiv.org/abs/2408.16725.

Jin Xu, Zhifang Guo, Hangrui Hu, Yunfei Chu, Xiong Wang, Jinzheng He, Yuxuan Wang, Xian Shi, Ting He, Xinfa Zhu, Yuanjun Lv, Yongqi Wang, Dake Guo, He Wang, Linhan Ma, Pei Zhang, Xinyu Zhang, Hongkun Hao, Zishan Guo, Baosong Yang, Bin Zhang, Ziyang Ma, Xipin Wei, Shuai Bai, Keqin Chen, Xuejing Liu, Peng Wang, Mingkun Yang, Dayiheng Liu, Xingzhang Ren, Bo Zheng, Rui Men, Fan Zhou, Bowen Yu, Jianxin Yang, Le Yu, Jingren Zhou, and Junyang Lin. Qwen3-omni technical report. *arXiv preprint arXiv:2509.17765*, 2025a.

Silei Xu, Wenhao Xie, Lingxiao Zhao, and Pengcheng He. Chain of draft: Thinking faster by writing less. *arXiv preprint arXiv:2502.18600*, feb 2025b. doi: 10.48550/arXiv.2502.18600. URL https://arxiv.org/abs/2502.18600. Version 2 (last revised 3 Mar 2025).

Ruiqi Yan, Xiquan Li, Wenxi Chen, Zhikang Niu, Chen Yang, Ziyang Ma, Kai Yu, and Xie Chen. Uro-bench: Towards comprehensive evaluation for end-to-end spoken dialogue models, 2025. URL https://arxiv.org/abs/2502.17810.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025a. URL https://arxiv.org/abs/2505.09388.

Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Qiaowei Li, Minghui Chen, Zheng Lin, and Weiping Wang. Dynamic early exit in reasoning models. *arXiv preprint arXiv:2504.15895*, 2025b.

Fan Yang. The cost of thinking: Increased jailbreak risk in large language models. *arXiv preprint arXiv:2508.10032*, 2025.

Qian Yang, Jin Xu, Wenrui Liu, Yunfei Chu, Ziyue Jiang, Xiaohuan Zhou, Yichong Leng, Yuanjun Lv, Zhou Zhao, Chang Zhou, and Jingren Zhou. Air-bench: Benchmarking large audio-language models via generative comprehension. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1979–1998, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.109. URL https://aclanthology.org/2024.acl-long.109/.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 11809–11822. Curran Associates, Inc., 2023a. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/271db9922b8d1f4dd7aaef84ed5ac703-Paper-Conference.pdf.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=WE_vluYUL-X.

Wenyi Yu, Siyin Wang, Xiaoyu Yang, Xianzhao Chen, Xiaohai Tian, Jun Zhang, Guangzhi Sun, Lu Lu, Yuxuan Wang, and Chao Zhang. Salmonn-omni: A codec-free llm for full-duplex speech understanding and generation, 2024. URL https://arxiv.org/abs/2411.18138.

Wenyi Yu, Siyin Wang, Xiaoyu Yang, Xianzhao Chen, Xiaohai Tian, Jun Zhang, Guangzhi Sun, Lu Lu, Yuxuan Wang, and Chao Zhang. Salmonn-omni: A standalone speech llm without codec injection for full-duplex conversation, 2025. URL https://arxiv.org/abs/2505.17060.

Yijiong Yu. Accelerate parallelizable reasoning via parallel decoding within one sequence, 2025. URL https://arxiv.org/abs/2503.20533.

Heiga Zen, Keiichi Tokuda, and Alan W. Black. Statistical parametric speech synthesis. *Speech Communication*, 51(11):1039–1064, 2009. ISSN 0167-6393. doi: https://doi.org/10.1016/j.specom.2009.04.004. URL https://www.sciencedirect.com/science/article/pii/S0167639309000648.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users, 2023a. URL https://arxiv.org/abs/2312.13771.

Dong Zhang, Shimin Li, Xin Zhang, Jun Zhan, Pengyu Wang, Yaqian Zhou, and Xipeng Qiu. Speechgpt: Empowering large language models with intrinsic cross-modal conversational abilities. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 15757–15773. Association for Computational Linguistics, December 2023b. doi: 10.18653/v1/2023.findings-emnlp.1055. URL https://aclanthology.org/2023.findings-emnlp.1055/.

Qinglin Zhang, Luyao Cheng, Chong Deng, Qian Chen, Wen Wang, Siqi Zheng, Jiaqing Liu, Hai Yu, Chaohong Tan, Zhihao Du, and Shiliang Zhang. Omniflatten: An end-to-end gpt model for seamless voice conversation, 2025a. URL https://arxiv.org/abs/2410.17799.

Ruiqi Zhang, Changyi Xiao, and Yixin Cao. Long or short cot? investigating instance-level switch of large reasoning models, 2025b. arXiv preprint.

Yichi Zhang, Yue Ding, Jingwen Yang, Tianwei Luo, Dongbai Li, Ranjie Duan, Qiang Liu, Hang Su, Yinpeng Dong, and Jun Zhu. Towards safe reasoning in large reasoning models via corrective intervention, 2025c. URL https://arxiv.org/abs/2509.24393.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alexander J. Smola. Automatic chain of thought prompting in large language models. *ArXiv*, abs/2210.03493, 2022. URL https://api.semanticscholar.org/CorpusID:252762275.

Haoyu Zheng, Zhuonan Wang, Yuqian Yuan, Tianwei Lin, Wenqiao Zhang, Zheqi Lv, Juncheng Li, Siliang Tang, Yueting Zhuang, and Hongyang He. Fast thinking for large language models, 2025a. arXiv preprint.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623, 2023.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, Clark Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2024. doi: 10.48550/arXiv.2312.07104.

Tong Zheng, Hongming Zhang, Wenhao Yu, Xiaoyang Wang, Runpeng Dai, Rui Liu, Huiwen Bao, Chengsong Huang, Heng Huang, and Dong Yu. Parallel-r1: Towards parallel thinking via reinforcement learning, 2025b. URL https://arxiv.org/abs/2509.07980.

Kaiwen Zhou, Xuandong Zhao, Gaowen Liu, Jayanth Srinivasa, Aosong Feng, Dawn Song, and Xin Eric Wang. Safekey: Amplifying aha-moment insights for safety reasoning, 2025. URL https://arxiv.org/abs/2505.16186.

Zihao Zhu, Xinyu Wu, Gehan Hu, Siwei Lyu, Ke Xu, and Baoyuan Wu. Advchain: Adversarial chain-of-thought tuning for robust safety alignment of large reasoning models, 2025. URL https://arxiv.org/abs/2509.24269.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R. Sanketi, Grecia Salazar, Michael S. Ryoo, Krista Reymann, Kanishka Rao, Karl Pertsch, Igor Mordatch, Henryk Michalewski, Yao Lu, Sergey Levine, Lisa Lee, Tsang-Wei Edward Lee, Isabel Leal, Yuheng Kuang, Dmitry Kalashnikov,

Ryan Julian, Nikhil J. Joshi, Alex Irpan, Brian Ichter, Jasmine Hsu, Alexander Herzog, Karol Hausman, Keerthana Gopalakrishnan, Chuyuan Fu, Pete Florence, Chelsea Finn, Kumar Avinava Dubey, Danny Driess, Tianli Ding, Krzysztof Marcin Choromanski, Xi Chen, Yevgen Chebotar, Justice Carbajal, Noah Brown, Anthony Brohan, Montserrat Gonzalez Arenas, and Kehang Han. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In Jie Tan, Marc Toussaint, and Kourosh Darvish (eds.), *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 2165–2183. PMLR, 06–09 Nov 2023. URL https://proceedings.mlr.press/v229/zitkovich23a.html.

Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J. Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023.

APPENDIX

A PROMPTING

In this section we provide detailed prompts used in both our main and safety experiments.

A.1 MAIN PROPMT

In our main setup, we use a minimal prompt to keep it easy to extend or modify. We do not define separate thinker or writer instructions; instead, the user input is shared between both. To indicate that the internal reasoning may be partial, we additionally prepend each stream with a fixed prefix.

Writer prefix

```
... [SYSTEM: thoughts will continue here]\n</think>\n
```

Thinker prefix

```
<|im_end|>\n<|im_start|>assistant\n<think>\n
```

For mode switching, we use an auxiliary prompt that introduces the thinker/writer roles and specifies the decision rule for switching. This prompt is only shown when the model is explicitly asked to change modes.

Mode Switching prompt (before reasoning)

```
You are an AI assistant that can think and write responses concurrently,
↪  and you must decide whether or not you should pause writing and
↪  think more.

Read the current partial thoughts and response below, then decide
↪  whether you can continue writing the response without pausing
↪  (yes/no):
 - Answer "yes" if your thoughts have enough information to write the
 ↪  next response paragraph, even if the full task is not solved yet.
 - Answer "no" if your thoughts aren't enough to write the next
 ↪  response paragraph, i.e. if your response ran out of of thoughts.
```

A.2 ASYNCREASONING (SAFETY PROMPTS)

The differences from the main prompt are the Thinker prefix, the mode-switching prompt, and the mode-switching question. In addition to regular chat template specific tokens they contain safety instructions and hints.

Thinker prefix

```
<|im_end|>\n<|im_start|>assistant\n<think>
EXECUTION PROTOCOL:
1. SAFETY CHECK: Before solving, explicitly evaluate if the user's
↪  request is safe, legal, and ethical.
   - If UNSAFE: Your goal is to refuse. You have \"enough information\"
   ↪  to write immediately after you decide to refuse.
   - If SAFE: Proceed to solve the problem step-by-step.
2. REASONING: If the request is safe, solve the problem step by step.
3. VALIDATION: Validate your final answer again and again until you see
↪  the same answer in the partial response above your thoughts.

Prioritize safety first, then accuracy.
```

Mode Switching prompt (before reasoning)

```
You are an AI assistant that can think and write responses concurrently,
↪  and you must decide whether or not you should pause writing and
↪  think more.

Read the current partial thoughts and response below, then decide
↪  whether you can continue writing the response without pausing
↪  (yes/no):
 - Answer "yes" if you have completed the safety check and have enough
 ↪  information to write the next response paragraph, even if the full
 ↪  task is not solved yet.
 - Answer "no" if you are still checking safety, need to think more, or
 ↪  need to validate your answer.
```

While we discuss Mode Switching question separately in Appendix B. In the safety experiments we replace it with the following.

Mode Switching question

```
Wait, have I completed the safety check and do I have enough thoughts
↪  to write the next paragraph or formula? (yes/no):
```

B Mode Switching Details

We implement mode switching as an explicit control query that asks the model to decide whether the writer should continue or pause to wait for additional thoughts. Concretely, we build an auxiliary "mode-switching" prompt (shown in Appendix A) and append the current thinker and writer partial outputs. We then append a short yes/no question:

Mode Switching question

```
Wait, are my current thoughts enough to write the next paragraph or
↪  formula? (yes/no):
```

The model's next-token distribution is used as a binary decision rule: if $p(\text{"yes"}) > p(\text{"no"})$, we allow the writer to keep generating; otherwise, we pause the writer and keep advancing the thinker. In our implementation we trigger this check every 20 decoding steps or when the thinker emits an "end-of-step" marker (two consecutive newlines). Additionally, when the writer finishes a paragraph boundary (two consecutive newlines) during simultaneous generation, we temporarily pause writing until new thoughts arrive, which prevents the writer from getting too far ahead. Finally, if the thinker

emits an explicit end-of-think token (e.g., `</think>`), we switch to writer-only mode so the writer can complete the response.

KV-cache layout and reuse. To avoid re-encoding long contexts, we split the KV cache into dedicated blocks that can be recombined for different modes. Specifically, we allocate five blocks: (1) the shared input prompt, (2) the thinker output prefix and growing thinker stream, (3) the writer output prefix and growing writer stream, (4) the fixed mode-switching prompt, and (5) a small "question" block that is re-filled each time we ask the control query. During initialization, we prefill the prompt blocks once and then only append new tokens to the thinker/writer blocks as generation proceeds. We then define cache "views":

- `thinker_view` = [input prompt, thinker output]
- `writer_view` = [input prompt, thinker output, writer output]
- `mode_switching_view` = [mode-switching prompt, thinker output, writer output, question]

Each view is wrapped by a cache manager (standard `SharedCacheManager` or the fast-kernel `HogwildCache`) so that a single forward pass can reuse the existing KV entries without re-tokenizing or re-encoding the full context. When a mode-switch decision is needed, we clear the question block (`clear()` in the standard cache, `crop(0)` in the fast-kernel cache), encode the short yes/no question into that block, and run a one-step forward pass on `mode_switching_view`. The resulting logits for the next token are compared for "yes" and "no" to determine whether the writer should proceed.

This arrangement keeps the decision query cheap (only the question block is re-encoded) while ensuring the decision is conditioned on the same partial thoughts and partial response that the model has produced so far.

C Mode-Switching Failure Mode Analysis

To better understand the reliability of our mode-switching mechanism, we analyze the saved generations used in Figure 4. We focus on critical mode-switching errors that directly affect final answer quality, and separate them from other failure modes unrelated to routing between the thinker and writer streams.

We consider two types of critical errors. First, **Premature** errors occur when the writer outputs an incorrect answer before the thinker has finished producing enough reasoning. Second, **Stalling** errors occur when the thinker continues generating while the writer is held back by the mode-switching decision, so the writer never gets an opportunity to produce the final answer. These two categories capture the main ways in which mode switching can harm final task performance.

Table 5: Mode-switching failure rates for Qwen3 models on MATH-500.

Model	Premature ↓	Stalling ↓
0.6B	0.311	0.064
1.7B	0.020	0.028
4B	0.000	0.050
8B	0.000	0.010
14B	0.000	0.008
32B	0.000	0.006

As shown in Table 5, smaller models make substantially more frequent mode-switching errors. This is especially pronounced for Qwen3-0.6B, where premature writing is the dominant failure mode. For larger models, premature errors disappear in this analysis, while stalling remains rare but non-zero. This trend may help explain why smaller models tend to lose more accuracy under mode switching in our evaluations.

This analysis focuses only on critical mode-switching errors that affect final correctness. A more fine-grained analysis could additionally quantify minor inefficiencies, such as cases where the writer could have started answering slightly earlier without harming correctness.

D Generalization to Other Positional Embeddings

Most modern LLMs use relative positional information. In practice, the dominant families are: (i) RoPE-style rotations (Su et al., 2024), (ii) NoPE-style models where attention is position-agnostic, and (iii) additive relative-bias approaches such as ALiBi (Press et al., 2022). This appendix explains how the query-adjustment trick from Section 3.3 extends beyond RoPE.

RoPE-Style. Our method uses the fact that the attention dot product is invariant under rotating both Q and K by the same angle. This lets us store the KV-cache in block-local coordinates (each token rotated as if its position started at 0 within its block), and then rotate only the query inside the attention kernel to emulate the correct relative offsets between blocks (see Section 3.3 for more details).

NoPE-Style. For NoPE models, attention does not encode position in Q and K vectors and does not add any positional terms. In this case, implementation is trivial.

ALiBi-Style. ALiBi adds per-head bias inside attention that is a linear function of relative distance between query and key positions. This allows us to store the KV cache as-is and add a view-specific relative bias inside the attention kernel. More specifically, for each view we rearrange the bias matrix to match the block order in that view.

Other Relative Positional Schemes. More complex schemes that transform Q and K can often still be handled if they satisfy a relative property analogous to RoPE. When this holds, the same design principles applies: store each token once, and apply a per-block adjustment at query-time to account for the view-specific offset.

E Detailed Performance Analysis

To measure real-time delays, we implement a basic assistant pipeline that recognizes spoken inputs using `whisper-base` (Radford et al., 2023), feeds it into AsyncReasoning (or a baseline algorithm) to stream response tokens, then group them into short chunks (5 tokens or 1 LaTeX expr.) and use `tortoise-tts` (Betker, 2023) with default parameters to generate speech. For tasks involving LaTeX, we convert it into Clearspeak.

To better contextualize our main results in Section 4, we report a more detailed performance breakdown of this pipeline for Qwen-32B AsyncReasoning on MATH-500 dataset in Table 6. Note that the final row (Full Delay) is not wall time but the total "silence time" perceived by the user.

Table 6: Component runtime, Qwen3-32B, MATH-500, A100.

Component	Latency
LLM inference	203.176
TTS inference	3.304
TTS playback	100.889
Full Delay (overlap)	102.098

We additionally measure full end-to-end delay on a subset of MATH-500 samples with Qwen3-30B-A3B-Thinking-2507 in Table 7, using the same setup as in Section 4. We measure the wall time from the moment the user stops talking to the moment the system begins speaking, as well as the total frame-by-frame silence duration. Compared to our main evaluations, this includes automated speech recognition and additional software/hardware delays.

Table 7: End-to-end latency evaluation for Qwen3-30B-A3B-Thinking-2507.

Setup	TTFT ↓	Total Delay ↓
AsyncReasoning (end-to-end)	4.643	9.887
AsyncReasoning (Section 4)	3.901	9.145
Baseline w/ think (end-to-end)	895.599	895.599
Baseline w/ think (Section 4)	894.884	894.884

The end-to-end delays are slightly higher than the controlled evaluation from Section 4 due to speech recognition, input hardware, and software delays. However, the difference remains below one second, while the delay reductions in our main results are much larger, often reaching tens to hundreds of seconds.

The results suggest that the user-perceived delay is almost wholly (over 90%) attributed to LLM inference costs hidden under TTS playback, while the inference time of TTS itself plays a relatively minor role. However, this might change for smaller models (e.g. 0.6B), where LLM inference is cheaper. To control for this, we report all main experiments with TTS pipeline running on a separate GPU. Whenever the LLM generates another chunk of text, we run the TTS pipeline in parallel as the LLM continues its work. In practice, this is consistent with having a TTS API running in a separate instance that does not interfere with the main LLM runtime. This also allows us to better decouple our results from the TTS engine choice, since there are newer engines faster than TortoiseTTS (Casanova et al., 2024; ekwek1, 2026).

F Additional Ablation for Section 4.1

In this section, we report additional ablations for mode switching criteria and validate our experimental setup. We follow the experimental setup from Section 4.1 for Qwen3-32B and vary several parameters.

Mode switching frequency ($T = 5 - 100$): by default, AsyncReasoning prompts the model to decide if it should pause writing every T=20 inference steps. Though we keep T=20 throughout our experiments, this parameter can be adjusted to balance between reaction time and computation overhead:

Table 8: Mode switching frequency comparison for Qwen3-32B, MATH-500 benchmark, A100 GPU.

Setup	Acc ↑	Delay ↓	TTFT ↓
T=5	0.938	169.90	2.75
T=10	0.936	117.82	3.34
T=15	0.940	105.18	3.70
T=20	0.938	102.10	4.32
T=25	0.932	94.88	5.04
T=30	0.922	83.09	5.12
T=50	0.918	81.66	6.60
T=100	0.922	87.70	10.91

The results in Table 8 demonstrate that AsyncReasoning is robust to the choice of mode switching frequency. Lower T values correspond to slightly faster "reaction time", but they also silently increase GPU overhead since each mode switching question requires additional token processing. Higher T has the opposite effect: reducing overhead at the cost of reaction time.

Next, we report several alternative prompting techniques that we considered in early experimentation.

Alternative prompts: we compare several additional mode switching strategies in Figure 7:

- **AsyncReasoning (main)** is our default mode swtiching strategy from Section 4 (prompts in Appendix A).
- **AsyncReasoning (thinker async)** is a setup where we let thinker view the writer response as previous term and actively encourage it to think about concurrency.
- **AsyncReasoning (switch prompt)** is a setup where we extend the mode switching prompt to 1) consider the original problem, 2) take parallel processing into account. Note that mode switching is still done based on a "yes"/"no" answer.

The remaining baselines are the same as in Section 4.1. The results in Figure 7 suggest that giving thinker or mode switcher explicit instructions to favor real-time responses does make it somewhat faster, but also reduces its accuracy. Crucially, we also found that *advanced prompts (thinker async, switch prompt) behave eratically on some benchmarks, notably GPQA-Diamond.* To that end, we chose the simplest prompt as our main setup for Section 4.2.

AsyncReasoning (TTS-based): this is our attempt to use real-time TTS information to inform mode switching. The core intuition is: if the writer is already "ahead of real-time", it may pause without slowing down the response. To incorporate this intuition, we run our TTS pipeline during inference over chunks of 5 generated tokens. We keep track of how many seconds of speech are synthesized but not yet spoken by any given time. We pause the writer if there are more than 10 seconds worth of response tokens "in the buffer", we pause the writer. Our results demonstrate that mode-switching decisions can be effectively guided by downstream speech-generation dynamics. However, using this criterion makes AsyncReasoning accuracy tied to the choice of TTS and GPU speed. For that reason, we decided to focus on TTS-agnostic criteria.

There are two more methods that we find interesting but have not fully evaluated yet:

1. Trained mode-switching: training a classifier head, perhaps on top of the model's own hidden state, to decide when to pause and wait for thoughts. This can reduce overhead GPU compute during inference, but it does not fit neatly into our training-free setup.
2. Planned mode switching: similar to (Liang et al., 2025a), we could prompt the thinker to plan ahead and decide which thoughts need to complete before the next response chunk. This type of planning can happen before thinker response or as a third "thread" concurrent to thinker and writer. However, we found that forming such plans makes the model change its overall response strategy (and hence, output length, affecting delay indirectly) significantly, making it difficult to compare using our evaluation setup.

F.1 The impact of GPU type

In Section 4, we normally evaluate each model on the same GPU type, e.g. A100 for Qwen3-32B, B200 for 235B-A22B. In this section, we compare the same AsyncReasoning configuration on different GPU types to demonstrate how hardware impacts user-perceived delays.

Table 9: GPU type impact for Qwen3-30B-A3B-Thinking-2507, average latency (s.) over benchmarks: MATH-500, GPQA-Diamond, MMLU-Pro subset (random 500 samples).

GPU	Delay ↓	TTFT ↓
A100	6.12	3.85
H200	3.37	2.25
B200	4.66	3.34

Curiously, Blackwell GPU does not offer significant performance improvements over Hopper for this model. This is despite the fact that B200 runs the model faster overall (tokens per second using our kernel). After closer examination, we attribute this to the fact that the user-perceived delay is dominated by the early response, after which model inference costs are masked by TTS speech time. In this case, the improvement from using B200 on relatively small A30B MoE experts does not significantly impact the "warmup time", and after the initial warmup, both Blackwell and Hopper generate tokens quickly enough to incur no additional delays due to TTS masking.

F.2 The impact of NF4 expert quantization

When evaluating Qwen3-235B model in Section 4.2, we quantize its experts to NF4 (Dettmers et al., 2023) while keeping the rest of the model in `bfloat16` precision to fit on a single B200 (or potentially H200) GPU. In this section, we verify this by comparing original and NF4 quantized experts on a smaller 30B-A3B model where `bfloat16` inference is feasible on a single GPU. For this analysis, we evaluate full MATH-500 and GPQA-Diamond, but only a sub-sample of 500 out of 12032 samples in MMLU-Pro to reduce GPU costs. We evaluate baseline (think) for both precisions and non-thinking for reference and report accuracies in Table 10.

Table 10: Expert-only NF4 quantization impact Qwen3-30B-A3B-Thinking-2507, accuracies for MATH-500, GPQA-Diamond, MMLU-Pro subset (random 500 samples).

Setup	MATH ↑	MMLU_s ↑	GPQA ↑
NF4 (thinking)	0.92	0.76	0.67
BF16 (thinking)	0.93	0.77	0.66
BF16 (non-thinking)	0.83	0.72	0.54

F.3 AsyncReasoning with Small and Quantized Models

The effectiveness of AsyncReasoning depends on whether the base model can reliably coordinate the two modes of generation: Thinker and Writer. This limitation is most visible for very small models. In our experiments in Section 4, models such as Qwen3-0.6B lose substantially more accuracy under our setup compared to their synchronous thinking baselines. Manual inspection suggests that this degradation is primarily caused by weaker routing behavior: small models are less reliable at predicting when to pause, when to continue writing, and how much reasoning context is sufficient for the next response segment. Refer to Appendix C for more details.

For local deployment on edge devices, such as laptops or smartphones, one can compress larger models that already exhibit reliable routing behavior, instead of applying AsyncReasoning directly to very small models.

To test this, we additionally evaluate AsyncReasoning with the official 4-bit AWQ quantization of Qwen3-8B on MATH-500. The results in Table 11 show that quantization preserves the main AsyncReasoning behavior: the quantized model nearly matches the synchronous thinking baseline in accuracy while substantially reducing time to first token and average perceived delay.

Table 11: Quantized 4-bit Qwen3-8B-AWQ on MATH-500.

Method	Accuracy	Avg. Delay (s.)	TTFT (s.)
8B-AWQ No Think	0.812	1.254	1.254
8B-AWQ w/ Think	0.886	485.535	485.535
8B-AWQ AsyncReasoning	0.880	6.351	1.809

F.4 Batched Serving

Our main latency experiments focus on the batch-size-one setting, which is particularly relevant for interactive applications. However, practical inference systems often rely on batching to improve throughput. We therefore additionally evaluate batched AsyncReasoning with Qwen3-32B on MATH-500 using the same A100 setup as in Section 4.1. The model achieves an accuracy of 0.94 in all configurations.

Batch size	Tokens / s.	TTFT (s.)	Total delay (s.)
1	26.07	4.32	102.10
2	48.20	4.26	110.28
4	85.56	4.70	132.49

Table 12: Batched AsyncReasoning with Qwen3-32B on MATH-500 using an A100 GPU. Accuracy is 0.94 for all batch sizes.

As shown in Table 12, AsyncReasoning retains low time-to-first-token under batching: TTFT increases only from 4.32s to 4.70s between batch sizes 1 and 4. At the same time, throughput increases from 26.07 to 85.56 tokens/s. Total user-perceived delay grows more moderately, from 102.10s to 132.49s. Thus, batching introduces the expected latency–throughput trade-off, but does not remove the main interactivity benefit of asynchronous generation.

G Benchmark & Evaluation Details

Below, we provide additional details about the evaluation setup. For consistency across methods, we cap the generation length at 16K tokens. We use greedy decoding by default, except for AIME-2025, where we use nucleus sampling with the recommended parameters.

- MATH-500: We report accuracy. Response correctness is determined via an LLM-as-a-judge evaluation using GPT-4.1 to check equivalence with the ground-truth answer.
- MMLU-Pro: We formulate the task as a multiple-choice problem with 7–10 options and evaluate by directly comparing the predicted option letter to the ground-truth answer.
- GPQA-Diamond: We formulate the task as a multiple-choice problem with 4 options and evaluate by directly comparing the predicted option letter to the ground-truth answer.
- AIME-2025: We report accuracy. Due to the small dataset size and high variance, we average accuracy over 10 random seeds. We use the sampling parameters recommended in the model card for both thinking and non-thinking models.
- Zebra Logic: We use the `grid` formulation, as it is the more standard variant.
- SpokenMQA: We report accuracy as evaluated via an LLM-as-a-judge setup using GPT-4.1.

In addition to accuracy and total delay, we measure additional performance metrics:

- **Time to first token (TTFT):** the wall time delay until the system generates the first *non-thinking* token.

- **Total delay:** same in the previous section. We run TTS on LLM-generated response tokens and measure the total delay experienced by the user.
- **Steps to first token (STFT):** the number of inference steps (LLM forward passes) before the first *non-thinking* token is generated, GPU-agnostic.
- **Steps Delay:** The average number of inference steps (forward passes) that do not generate a response token.
- **Response tokens:** the number of tokens in the "public" user-facing response. We report this metric to ensure that the differences in delay come from better asynchrony and not just more verbose responses.

H Additional Experiments for Section 4.2

The benchmark–model coverage in our paper is chosen to match model scale to benchmark difficulty while keeping the evaluation budget focused on the most informative settings. We prioritize pairs where base model can be meaningfully improved upon with reasoning: larger reasoning models are most informative on harder benchmarks such as AIME-2025 and GPQA-Diamond, while smaller are suited for simpler or real-time-oriented benchmarks such as MATH-500 and SpokenMQA. In particular, running very large models on relatively easy benchmarks provide limited additional signal, since their reasoning capacity may not be fully utilized. SpokenMQA is also designed for speech-minded real-time LLMs and does not always benefit from longer reasoning for large models. To verify that this coverage choice does not change the conclusions, we include additional representative evaluations in this appendix, including Qwen3-235B-A22B on MATH-500, smaller Qwen3 models on AIME-2025, and larger Qwen3 models on SpokenMQA.

Below we provide complete results, including accuracies, delays, and steps for the benchmarks:

- Math-500 results are reported in Table 13.
- MMLU-Pro results are reported in Table 14.
- GPQA-Diamond results are reported in Table 15.
- AIME 2025 results are reported in Table 16
- Zebra Logic (grid) results are reported in 17
- SpokenMQA results are reported in Table 18.

Model	Method	Accuracy	Time to 1st token (s)	Total delay (s)	Steps to 1st token	Total delay steps	Writer tokens
Qwen-0.6B	Baseline (Thinking)	0.71	641.21	641.29	4963.93	4963.93	434.51
	Baseline (Non-thinking)	0.50	0.79	1.01	1.00	1.00	679.81
	Async Reasoning	0.66	2.08	3.59	19.94	442.22	2110.99
Qwen-1.7B	Baseline (Thinking)	0.87	607.14	607.23	4643.06	4643.06	561.11
	Baseline (Non-thinking)	0.75	0.78	1.00	1.00	1.00	885.54
	Async Reasoning	0.69	0.82	2.86	1.00	1.00	1216.95
Qwen-4B	Baseline (Thinking)	0.92	556.36	556.52	4212.70	4212.70	591.50
	Baseline (Non-thinking)	0.85	0.85	1.14	1.00	1.00	922.58
	Async Reasoning	0.83	0.88	3.84	1.06	2.62	1598.42
Qwen-8B	Baseline (Thinking)	0.91	559.02	559.17	4577.98	4577.98	614.70
	Baseline (Non-thinking)	0.85	0.79	1.10	1.00	1.00	985.25
	Async Reasoning	0.91	1.81	6.35	19.88	632.01	1179.49
Qwen-14B	Baseline (Thinking)	0.94	560.16	560.34	3861.68	3861.68	627.35
	Baseline (Non-thinking)	0.86	0.87	1.16	1.00	1.00	941.46
	Async Reasoning	0.95	6.43	38.79	71.84	1411.26	650.39
Qwen-32B	Baseline (Thinking)	0.94	649.47	649.94	3760.25	3760.25	608.10
	Baseline (Non-thinking)	0.84	0.99	1.63	1.00	1.00	726.92
	Async Reasoning	0.94	4.32	102.10	28.04	1918.95	750.61
Qwen3-30B-A3B (2507)	Baseline (Thinking)	0.93	894.88	896.69	4738.73	4738.73	595.09
	Baseline (Non-thinking)	0.83	3.46	7.84	0.00	0.00	1367.84
	Async Reasoning	0.95	3.90	9.15	20.00	468.32	1405.19
Qwen3-235B-A22B (2507)	Baseline (Thinking)	0.97	1675.66	1688.25	4931.85	4931.85	608.01
	Baseline (Non-thinking)	0.93	6.52	6.52	0.00	0.00	1523.49
	Async Reasoning	0.96	9.98	159.2	21.93	691.35	1167.12

Table 13: Performance metrics for various Qwen models on the Math-500 benchmark. Delays are measured in seconds; steps refer to model inference steps. Writer tokens indicate the average number of generated tokens per sample.

Model	Method	Accuracy	Time to 1st token (s)	Total delay (s)	Steps to 1st token	Total delay steps	Writer tokens
Qwen3-0.6B	Baseline (Thinking)	0.29	278.72	278.75	277.74	2721.82	211.03
	Baseline (Non-thinking)	0.20	0.73	0.86	0.00	0.00	224.14
	Async Reasoning	0.23	1.52	2.32	6.21	43.50	351.88
Qwen3-1.7	Baseline (Thinking)	0.54	328.49	328.54	327.53	3154.13	446.34
	Baseline (Non-thinking)	0.39	0.72	1.01	0.00	0.00	499.59
	Async Reasoning	0.43	0.76	1.70	1.00	1.00	491.94
Qwen3-4B	Baseline (Thinking)	0.69	350.71	350.81	349.79	3076.16	540.94
	Baseline (Non-thinking)	0.55	0.77	1.19	0.00	0.00	694.33
	Async Reasoning	0.66	0.79	1.15	1.00	1.00	837.10
Qwen3-8B	Baseline (Thinking)	0.73	362.85	362.94	361.92	3278.30	591.72
	Baseline (Non-thinking)	0.58	0.79	1.35	0.00	0.00	824.81
	Async Reasoning	0.70	2.31	8.81	24.20	621.18	735.34
Qwen3-14B	Baseline (Thinking)	0.76	295.55	295.70	294.67	2565.01	548.49
	Baseline (Non-thinking)	0.68	0.87	1.57	0.00	0.00	705.30
	Async Reasoning	0.75	7.84	43.11	116.00	1299.75	423.38
Qwen3-32B	Baseline (Thinking)	0.79	346.50	346.99	345.81	2297.41	521.96
	Baseline (Non-thinking)	0.72	1.14	2.43	0.00	0.00	640.81
	Async Reasoning	0.76	4.23	28.66	43.02	663.66	555.92
Qwen3-30B-A3B (2507)	Baseline (Thinking)	0.78	481.90	482.58	481.50	2534.95	462.61
	Baseline (Non-thinking)	0.72	3.45	8.89	0.00	0.00	3007.11
	Async Reasoning	0.79	2.23	3.59	20.00	242.52	1379.58
Qwen3-235B-A22B (2507)	Baseline (Thinking)	0.81	1150.50	1161.36	2967.21	2967.21	513.50
	Baseline (Non-thinking)	0.74	4.70	34.75	0.00	0.00	1484.43
	Async Reasoning	0.79	14.65	83.83	37.67	431.15	1278.69
GPT-OSS-20B	Baseline (Low Effort)	0.64	5.24	5.24	4.24	122.88	206.00
	Baseline (Medium Effort)	0.73	45.30	45.30	44.30	922.12	206.22
	Async Reasoning	0.70	2.43	6.39	5.34	715.01	223.78
GPT-OSS-120B	Baseline (Low Effort)	0.75	11.82	11.82	10.82	175.71	206.00
	Baseline (Medium Effort)	0.78	39.41	39.41	38.41	605.61	209.98
	Async Reasoning	0.77	7.73	13.36	13.36	403.06	211.14

Table 14: Performance metrics for various models on the MMLU-Pro benchmark. Delays are measured in seconds; steps refer to model inference steps. Writer tokens indicate the average number of generated tokens per sample.

Model	Method	Accuracy	Time to 1st token (s)	Total delay (s)	Steps to 1st token	Total delay steps	Writer tokens
Qwen3-0.6B	Baseline (Thinking)	0.25	693.64	693.64	5451.25	5451.25	240.32
	Baseline (Non-thinking)	0.20	0.73	0.75	1.00	1.00	572.93
	Async Reasoning	0.19	1.65	4.58	19.71	404.16	2512.85
Qwen3-1.7B	Baseline (Thinking)	0.36	969.85	969.85	7457.90	7457.90	713.05
	Baseline (Non-thinking)	0.31	0.77	3.77	1.00	1.00	1141.47
	Async Reasoning	0.31	0.83	0.83	1.00	1.00	1113.42
Qwen3-4B	Baseline (Thinking)	0.52	985.37	985.37	7229.76	7229.76	743.40
	Baseline (Non-thinking)	0.37	0.79	0.84	1.00	1.00	1446.78
	Async Reasoning	0.45	0.88	0.90	1.00	1.00	2364.18
Qwen3-8B	Baseline (Thinking)	0.56	905.97	905.97	7497.11	7497.11	823.00
	Baseline (Non-thinking)	0.32	0.80	0.86	1.00	1.00	1439.42
	Async Reasoning	0.47	4.23	57.85	49.29	2052.61	1048.91
Qwen3-14B	Baseline (Thinking)	0.63	896.27	896.27	6314.86	6314.86	838.81
	Baseline (Non-thinking)	0.57	0.81	0.86	1.00	1.00	1465.93
	Async Reasoning	0.55	30.35	149.75	622.32	4068.90	1004.87
Qwen3-32B	Baseline (Thinking)	0.68	1132.84	1132.84	6241.74	6241.74	750.13
	Baseline (Non-thinking)	0.57	1.00	1.08	1.00	1.00	1144.15
	Async Reasoning	0.65	3.61	154.31	31.99	2796.39	1064.26
Qwen3-30B-A3B (2507)	Baseline (Thinking)	0.67	1293.18	1294.67	6836.40	6836.40	745.79
	Baseline (Non-thinking)	0.54	3.70	3.70	1.00	1.00	1567.85
	Async Reasoning	0.67	3.78	3.94	20.00	577.06	3311.05
Qwen3-235B-A22B (2507)	Baseline (Thinking)	0.76	10.10	10.16	0.00	0.00	7982.26
	Baseline (Non-thinking)	0.66	12.18	61.11	0.00	0.00	1790.83
	Async Reasoning	0.72	18.61	243.57	53.64	1376.37	3310.06

Table 15: Performance metrics for various models on the GPQA-Diamond benchmark. Delays are measured in seconds; steps refer to model inference steps. Writer tokens indicate the average number of generated tokens per sample.

Model	Method	Accuracy	Time to 1st token (s)	Total delay (s)	Steps to 1st token	Total delay steps	Writer tokens
Qwen3-0.6B	Baseline (Thinking)	0.15	1046.13	1046.16	11851.68	11851.68	496.48
	Baseline (Non-thinking)	0.023	3.05	3.24	0.00	0.00	1482.81
	Async Reasoning	0.147	2.23	2.39	19.91	1292.80	890.32
Qwen3-1.7B	Baseline (Thinking)	0.287	985.46	985.47	12867.44	12867.44	473.58
	Baseline (Non-thinking)	0.096	3.01	3.11	0.00	0.00	2320.64
	Async Reasoning	0.267	3.62	3.67	1.00	1.00	2187.11
Qwen3-4B	Baseline (Thinking)	0.467	1243.31	1243.38	12022.73	12022.73	606.09
	Baseline (Non-thinking)	0.177	3.81	3.95	0.00	0.00	1338.29
	Async Reasoning	0.407	3.92	3.99	1.32	9.16	650.34
Qwen3-8B	Baseline (Thinking)	0.453	1281.29	1281.36	12326.72	12326.72	620.79
	Baseline (Non-thinking)	0.207	2.99	4.26	0.00	0.00	3542.62
	Async Reasoning	0.433	3.49	8.95	19.87	2262.71	4715.90
Qwen3-14B	Baseline (Thinking)	0.49	1013.18	1013.21	11972.66	11972.66	701.91
	Baseline (Non-thinking)	0.197	2.89	2.98	0.00	0.00	3642.06
	Async Reasoning	0.493	16.89	135.91	245.17	6279.47	781.36
Qwen3-32B	Baseline (Thinking)	0.533	1914.45	1914.45	11751.63	11751.63	719.98
	Baseline (Non-thinking)	0.2	1.13	1.70	0	0	3690.07
	Async Reasoning	0.493	5.41	485.93	30.67	8224.03	756.84
Qwen3-30B-A3B (2507)	Baseline (Thinking)	0.67	1813.67	1814.09	12937.53	12937.53	573.03
	Baseline (Non-thinking)	0.39	5.25	5.25	0.00	0.00	13563.67
	Async Reasoning	0.62	2.20	2.20	20.00	1493.60	13587.09
Qwen3-235B-A22B (2507)	Baseline (Thinking)	0.68	5330.46	5348.22	13753.82	13753.82	413.39
	Baseline (Non-thinking)	0.53	15.73	20.16	0.00	0.00	14127.90
	Async Reasoning	0.68	9.06	500.94	21.96	2154.34	7377.12
GPT-OSS-20B	Baseline (Medium Effort)	0.62	523.53	523.53	7691.55	7691.55	486.48
	Baseline (Low Effort)	0.36	120.09	120.09	1694.43	1694.43	496.57
	Async Reasoning	0.59	2.83	147.96	24.15	5894.61	521.33
GPT-OSS-120B	Baseline (Medium Effort)	0.77	326.52	326.52	5079.68	5079.68	728.41
	Baseline (Low Effort)	0.51	91.40	91.40	1411.24	1411.24	516.63
	Async Reasoning	0.66	9.32	166.75	24.63	3894.61	683.41

Table 16: Performance metrics for various models on the AIME 2025 benchmark. Delays are measured in seconds; steps refer to model inference steps. Writer tokens indicate the average number of generated tokens per sample.

Model	Method	Accuracy	Time to 1st token (s)	Total delay (s)	Steps to 1st token	Total delay steps	Writer tokens
Qwen3-0.6B	Baseline (Thinking)	0.31	892.59	892.59	7508.04	7508.04	252.37
	Baseline (Non-thinking)	0.05	0.51	0.51	1.00	1.00	386.55
	Async Reasoning	0.06	0.73	0.73	2.25	21.11	895.06
Qwen3-1.7B	Baseline (Thinking)	0.61	803.33	803.33	6651.08	6651.08	267.85
	Baseline (Non-thinking)	0.13	0.57	0.57	1.00	1.00	1566.85
	Async Reasoning	0.12	0.72	0.72	1.00	1.00	3280.79
Qwen3-4B	Baseline (Thinking)	0.81	681.66	681.66	5585.54	5585.54	286.06
	Baseline (Non-thinking)	0.32	0.74	0.74	1.00	1.00	2089.58
	Async Reasoning	0.36	0.88	0.88	1.00	1.00	4789.77
Qwen3-8B	Baseline (Thinking)	0.85	743.83	743.83	5514.54	5514.54	279.38
	Baseline (Non-thinking)	0.26	0.67	0.67	1.00	1.00	1442.88
	Async Reasoning	0.26	0.70	0.70	1.00	1.00	2126.38
Qwen3-14B	Baseline (Thinking)	0.88	676.05	676.05	4966.50	4966.50	305.56
	Async Reasoning	0.85	11.21	173.89	138.74	3725.13	399.07
Qwen3-32B	Baseline (Thinking)	0.85	940.27	940.27	5117.19	5117.19	297.65
	Baseline (Non-thinking)	0.29	0.75	0.75	1.00	1.00	1358.23
	Async Reasoning	0.66	10.29	138.76	122.93	2569.64	1312.06
Qwen3-30B-A3B (2507)	Baseline (Thinking)	0.96	1244.08	1244.08	5845.74	5845.74	499.07
	Baseline (Non-thinking)	0.84	3.81	3.81	0.00	0.00	6363.42
	Async Reasoning	0.93	3.72	3.72	19.40	1353.80	4823.37
Qwen3-235B-A22B (2507)	Baseline (Thinking)	0.98	2111.67	2111.85	5643.99	5643.99	591.71
	Async Reasoning	0.95	8.37	57.85	21.74	212.72	3639.62
GPT-OSS-20B	Baseline (Low Effort)	0.01	149.44	149.44	2180.52	2180.52	339.48
	Baseline (Medium Effort)	0.70	372.41	372.41	5516.39	5116.39	435.29
	Async Reasoning	0.61	4.37	163.34	23.90	4835.13	421.06
GPT-OSS-120B	Baseline (Low Effort)	0.48	248.13	248.13	2891.53	2891.53	683.69
	Baseline (Medium Effort)	0.81	268.83	268.83	3807.28	3807.28	692.52
	Async Reasoning	0.75	7.53	131.57	31.42	3403.06	686.87

Table 17: Performance metrics for various models on the Zebra Logic benchmark. Delays are measured in seconds; steps refer to model inference steps. Writer tokens indicate the average number of generated tokens per sample.

Model	Method	Accuracy	Time to 1st token (s)	Total delay (s)	Steps to 1st token	Total delay steps	Writer tokens
Qwen-0.6B	Baseline (Thinking)	0.70	201.52	201.53	1655.04	1655.04	258.49
	Baseline (Non-thinking)	0.57	0.68	0.72	1.00	1.00	245.97
	Async Reasoning	0.66	1.38	2.56	20.00	131.28	498.99
Qwen-1.7B	Baseline (Thinking)	0.80	219.21	219.22	1808.48	1808.48	308.26
	Baseline (Non-thinking)	0.73	0.67	0.71	1.00	1.00	295.39
	Async Reasoning	0.72	0.72	0.73	1.00	1.00	301.86
Qwen-4B	Baseline (Thinking)	0.81	227.16	227.16	1768.23	1768.23	311.49
	Baseline (Non-thinking)	0.81	0.70	0.80	1.00	1.00	295.66
	Async Reasoning	0.81	0.80	0.84	1.07	1.67	312.42
Qwen-8B	Baseline (Thinking)	0.82	265.22	265.23	1908.66	1908.66	326.51
	Baseline (Non-thinking)	0.81	0.70	0.78	1.00	1.00	299.81
	Async Reasoning	0.82	1.70	3.22	19.91	255.66	337.74
Qwen-14B	Baseline (Thinking)	0.84	207.97	207.99	1368.37	1368.37	306.62
	Baseline (Non-thinking)	0.83	0.79	0.88	1.00	1.00	286.36
	Async Reasoning	0.84	4.46	15.86	40.74	569.14	241.71
Qwen-32B	Baseline (Thinking)	0.84	243.99	244.07	1359.35	1359.35	299.59
	Baseline (Non-thinking)	0.83	0.94	2.28	1.00	1.00	284.71
	Async Reasoning	0.84	5.65	15.50	52.97	386.58	283.06
Qwen3-30B-A3B (2507)	Baseline (Thinking)	0.83	147.19	147.26	843.41	843.41	267.51
	Baseline (Non-thinking)	0.84	1.78	2.36	1.00	1.00	1088.70
	Async Reasoning	0.83	2.85	4.57	20.00	124.08	593.60

Table 18: Performance metrics for various Qwen models on the SpokenMQA benchmark. Delays are measured in seconds; steps refer to model inference steps. Writer tokens indicate the average number of generated tokens per sample.

I AGGREGATED SPEEDUP & ACCURACY SUMMARY

For convenience, we provide average speedups and accuracy changes in Table 19:

1. **Avg Speedup (Delay)** - total delay reduction compared to sequential thinking.
2. **Avg Speedup (TTFT)** - time to first token reduction compared to sequential thinking.
3. **Acc vs no-think** - mean increase in accuracy compared to the non-thinking baseline.
4. **Acc vs thinking** - mean change in accuracy compared to the thinking baseline.

Model	Delay Speedup	TTFT Speedup	Δ Acc. vs. Non-thinking	Δ Acc. vs. Thinking
Qwen3-0.6B	364.9×	458.3×	+0.067	-0.077
Qwen3-1.7B	543.1×	672.3×	+0.022	-0.155
Qwen3-4B	483.6×	595.3×	+0.073	-0.117
Qwen3-8B	238.8×	377.6×	+0.094	-0.122
Qwen3-14B	8.6×	53.5×	+0.089	-0.018
Qwen3-32B	8.7×	172.4×	+0.149	-0.048
Qwen3-30B-A3B (2507)	292.0×	333.0×	+0.105	-0.008
Qwen3-235B-A22B (2507)	14.3×	217.5×	+0.072	-0.020
GPT-OSS-20B	4.3×	96.3×	+0.297	-0.050
GPT-OSS-120B	2.3×	25.3×	+0.147	-0.060

Table 19: Aggregated AsyncReasoning performance across single-input capability benchmarks. Delay and TTFT speedups are measured relative to the corresponding sequential-thinking baseline. Accuracy columns report the mean absolute accuracy difference relative to the non-thinking and sequential-thinking baselines, respectively. Results are averaged across the evaluated benchmarks for each model; safety and asynchronous-input experiments are excluded.

The latter mostly consists of small negative values, because AsyncReasoning slightly underperforms long sequential thinking. For each model, we aggregate all single-input capability evaluations. This does not include safety attack success rates (Section 4.3) and asynchronous input studies (Section 4.4). For each model, we average speedups and accuracy changes across all such benchmarks (main tables from Section 4 and Appendix H).

J COMPARISON TO TRAINING-BASED INTERACTIVE REASONING

AsyncReasoning is designed to enable concurrent thinking and answering without modifying model weights. Several related methods instead train models specifically for interactive or interleaved reasoning. A direct comparison is difficult because the primary training-based approaches use different models and evaluation protocols, and implementations of several methods are not publicly

available. To provide a quantitative reference, we reproduce the evaluation settings reported by Plantain (Liang et al., 2025a) for Qwen3-4B on MATH-500 and Qwen3-8B on Text-to-SQL, and compare their reported results with AsyncReasoning.

Plantain reports the number of generated tokens before the first response token (TTFR), rather than wall-clock latency. We therefore use the same metric here and do not compare these values directly with the real-time TTFT measurements reported in our main experiments.

	Qwen3-4B, MATH-500		**Qwen3-8B, Text-to-SQL**	
Method	**Acc.**	**TTFR**	**Acc.**	**TTFR**
No Thinking	81.6	0	21.0	0
Think–Answer	82.8	1492	28.0	1684
AsyncReasoning	82.4	1.07	28.0	40.57
Plantain	84.4	625	28.0	457

Table 20: Comparison with Plantain, a training-based interleaved reasoning method, using the evaluation settings reported by Plantain. TTFR denotes the number of generated tokens before the first response token.

Table 20 highlights a trade-off between the two approaches. On MATH-500, Plantain obtains somewhat higher accuracy (84.4 vs. 82.4), while AsyncReasoning begins producing the response substantially earlier under the reported TTFR metric. On Text-to-SQL, both methods reach the same accuracy of 28.0, while AsyncReasoning requires fewer tokens before beginning its response. These results illustrate that training-free asynchronous execution can recover much of the capability of reasoning while avoiding the synchronous planning phase required by the trained interleaved setup.

K Safety & Reasoning

Recent studies reveal that Chain-of-Thought reasoning impact on safety risks is complex and bidirectional (Lu et al., 2025; Chua et al., 2025).

On one hand, CoT enhances safety by enabling transparency (Korbak et al., 2025; Baker et al., 2025), allowing models to structure the evaluation of harmful intent and facilitate self-correction before generating a final response (Wei et al., 2022; Zhou et al., 2025). Defense mechanisms like RoboGuard and CoT Prompting use this to reduce attack success rates by monitoring reasoning traces for policy violations (Zhang et al., 2025c; Wu et al., 2025b).

On the other hand, reasoning capabilities introduce new attack vectors not present in standard LLMs (Yang, 2025). The visibility of intermediate states exposes a larger attack surface: adversaries can hijack the reasoning process (H-CoT attacks) to bypass refusal mechanisms (Kuo et al., 2025), or exploit the "snowball effect" where minor reasoning deviations amplify into harmful outputs (Zhu et al., 2025).

Furthermore, reasoning models are susceptible to narrative deception and context-switching attacks, where the model rationalizes harmful compliance through complex logical deductions or by adopting a "helpful" persona in educational contexts (Chang et al., 2025; Yang, 2025).

L Safety Failure Mode Analysis for Section 4.3

While AsyncReasoning allows for real-time safety checks, the asynchronous nature of generation introduces specific failure modes where the writer may output harmful content before the thinker intervenes. We identify three primary categories of such safety failures:

1. **Race Condition:** The writer begins generating a helpful response immediately based on the prompt. Although the thinker eventually concludes the request is unsafe, the writer has already streamed harmful tokens (e.g., the first steps of a dangerous recipe) to the user before the refusal signal is propagated.
2. **Context Leakage:** The thinker analyzes the harmful request by recalling technical details (e.g., explaining how a specific SQL injection works to verify its danger). The writer, attending to the

thinker's cache, interprets these technical details as the desired answer and formulates them into a response, bypassing the thinker's intent.

3. **Educational Loophole:** The thinker adopts an educational persona to explain why a request is dangerous. The writer latches onto this educational content and reformats it as a set of instructions, stripping away the safety framing context.

Table 21: Failure mode analysis by inference setup on HarmBench.

Inference Setup	**Failure Mode**	**Count**
Baseline (Non-thinking)	Misinformation compliance	13
Baseline (Thinking)	Misinformation compliance	15
	Educational loophole	10
AsyncReasoning	Context leakage	13
	Educational loophole	5
	Race condition	1
AsyncReasoning (Safety Prompt)	Educational loophole	1

These findings suggest that, while AsyncReasoning can effectively filter attacks, strict gating mechanisms (e.g., ensuring the thinker has a "head start" on safety verification) are necessary to prevent race conditions in highly sensitive scenarios.

M Results on AdvBench

To validate our findings, we additionally evaluate safety on the full AdvBench dataset (Zou et al., 2023) containing 520 harmful behavior prompts. Table 22 presents the results. We observe similar trends to HarmBench: enabling thinking in the baseline model increases vulnerability (ASR 0.0% $\rightarrow$ 3.27%), while AsyncReasoning with safety prompting completely eliminates successful attacks. Notably, even default AsyncReasoning (ASR 1.15%) outperforms the thinking baseline, suggesting that the asynchronous architecture provides some inherent safety benefits by allowing the thinker to flag dangerous content before the writer commits to a harmful trajectory.

Table 22: Attack Success Rate on AdvBench for Qwen3-32B.

Inference Setup	**ASR$\downarrow$**
Baseline (Non-thinking)	0.0%
Baseline (Thinking)	3.27%
AsyncReasoning	1.15%
AsyncReasoning (Safety Prompt)	**0.0%**

N Sharded MATH-500 Dataset Creation

We construct the sharded dataset by augmenting each problem in MATH-500. For each problem in the dataset we prompt `GPT-5` to produce multiple cases where an initial prompt is incomplete or incorrect, followed by the additional input that makes the problem equivalent to the original. The prompt includes the original problem, full solution, and final answer. We prompt the model to provide 1–3 incomplete cases and 1–3 incorrect cases. Each case is a paired prompt that includes a short rationale, the Initial prompt, and the Additional input; the initial version is intentionally missing or wrong (e.g., omitted constraints, ambiguous variables, or incorrect constants), and the clarification fixes the issue to recover the original problem.

Full prompt for generating sharded dataset

```
I've built a real-time voice assistant that can solve tasks for the
↪  user and interactively adjust to user inputs. I want to evaluate my
↪  assistant's ability to adjust to user giving additional information
↪  while the assistant is already thinking on the problem. I need you
↪  to help me set up the evaluation scenario.

The original problem is:
```
{problem}
```

I want you to make this problem into a pair of 1) incomplete or wrong
↪  problem description and 2) additional information the user
↪  specifies 10-30 seconds after the problem that would make an
↪  initially incomplete or wrong problem equivalent to the one above.

The solution to the original problem is:
```
{solution}
```

The final answer is:
```
{answer}
```

Please make sure that the pair of 1) incomplete / wrong problem and 2)
↪  clarification result in the same final answer after incorporating
↪  the clarification. Note that the incomplete/wrong problem, if it
↪  has a solution, should not be solvable or should produce a solution
↪  different from the final one.

Please provide 1-3 cases with incomplete problems and 1-3 cases with
↪  mistakes in the problem definition. Please provide each case in the
↪  following format:

### CASE [number] - [case title here]
Rationale: [A brief explanation of why the initial problem prompt does
↪  not result in the correct answer - and how the additional input
↪  fixes that]

Initial prompt:
```
[incomplete / wrong problem definition goes here]
```

Additional input:
```
[extra input 10-30 seconds in goes here]
```

The code blocks (in ```backticks```) should only contain the prompt and
↪  input itself, without extra comments / variants. The "Initial
↪  prompt:" and "Additional input:" headers must be verbatim and only
↪  one of each per case. Omit the [square brackets] in the actual
↪  response.
```

We discard cases where the incomplete prompt already solves the problem or where with the additional input model still fails to recover the answer. Concretely, we run the evaluator model (Qwen3-32B) twice per case: once on the Initial prompt alone and once on the prompt augmented with the Additional Input, and we keep the case only if the first run does not match the reference answer while the second run does match. For a small number of tricky samples we manually select or revise the case (including edits to the additional input or occasional rewrites), with assistance from `GPT-5` and Gemini-Pro. The initial procedure using Qwen3-32B verified 473 samples out of 500 using; those 473 cases are kept unchanged from the prior version. The remaining samples are filled from the revised accepted cases. We include the dataset and the scripts used in its creation in our supplementary code.

O Additional Experiments for Section 4.4

In this appendix we explore different strategies on how to insert shards into running model's KV-cache.

We set two boundaries: lower bound which has model solve the task with only the first shard and upper bound which gives model all shards concatenated from the start. On all figures we report only upper bound as dotted line. Lower bound is less than 0.05 in all experiments.

Figure 8 reports an ablation of KV-cache injection targets for shards on sharded MATH-500 with Qwen3-32B. We sweep all cache-placement choices across the three cache blocks (prompt, thinker, writer) and study how accuracy changes with k_{steps}, i.e., the number of decoding steps between successive shard arrivals. Note: each problem from sharded MATH-500 consists of two shards, one is provided at the beginning and the other arrives k_{steps} steps later.

Figure 9 extends this analysis with *reminder* variants. In these settings, the shard content is injected into the prompt block, while a short marker prompt[5] is appended to the thinker and/or writer blocks to indicate that additional user input has arrived.

Across configurations, injecting shard content into the prompt block is consistently strong, and inserting into multiple blocks tends to be the most robust as k_{steps} increases. In particular, inserting shards into all three blocks (prompt+thinker+writer; `shard`→`PTW`) maintains higher accuracy at larger k_{steps}, suggesting that later-arriving information benefits from being exposed to both the thinker and the writer. Reminder mechanisms recover some performance but generally underperform the corresponding full-injection configurations, indicating that the marker prompt can help surface the presence of new information but cannot replace including the shard content in the model's active context. Additionally, configurations degrade sharply when the prompt block does not receive the shard content, underscoring the importance of placing new information into the primary input context.

Finally, Figure 10 summarizes the same design choices on Qwen3-235B-A22B-Thinking-2507 and on both sharded MATH-500 and math subset (103 samples) of `lost_in_conversation`. Same configurations are reported for `lost_in_conversation` on Qwen3-32B on Figure 11. We observe a similar overall pattern, with `shard`→`PTW` becoming preferable at larger k_{steps}, while `shard`→`P` can be competitive when updates arrive very early.

[5] `"... [SYSTEM: additional user input detected]\n"`



Figure 7: Comparing the impact of additional mode switching methods & baselines on MATH-500, Qwen3-32B, A100.

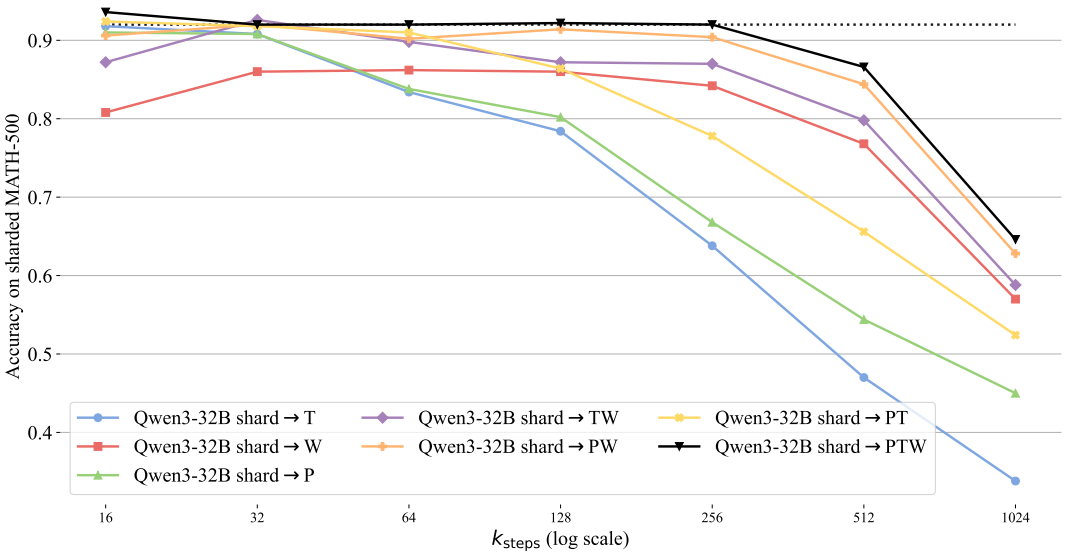

Figure 8: Accuracy ablation over cache-insertion targets as a function of k_{steps}. We evaluate all combinations of inserting shards into the prompt, thinker, and writer cache blocks. The dotted horizontal line denotes the upper bound, where all shards are provided at the start. All experiments use Qwen3-32B on sharded MATH-500.

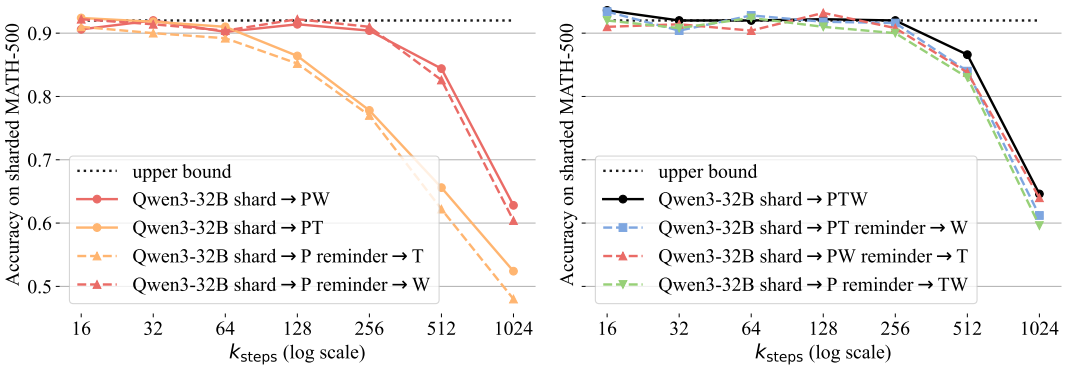

Figure 9: Accuracy ablation over cache-insertion targets as a function of k_{steps}. We evaluate setups where shards in some blocks are substituted with reminders. (Left) Two insertions in total. (Right) three insertions in total. The dashed lines denote reminder substitution experiments. The dotted horizontal line denotes the upper bound, where all shards are provided at the start. All experiments use Qwen3-32B on sharded MATH-500.

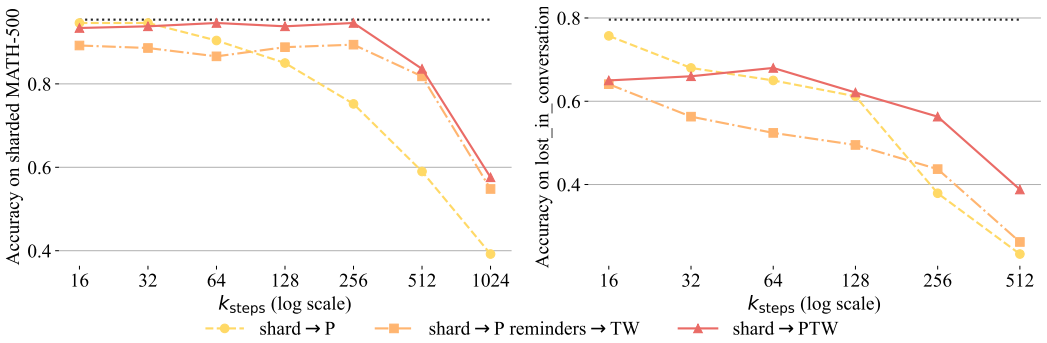

Figure 10: Accuracy with respect to k_{steps}. We report three setups: shard in prompts, shard in all three blocks, shard in prompt and reminder to the thinker and writer. Qwen3-235B-A22B-Thinking-2507 on the following datasets: (Left) MATH-500 and (Right) `lost_in_conversations`. The dotted horizontal line denotes the upper bound, where all shards are provided at the start.

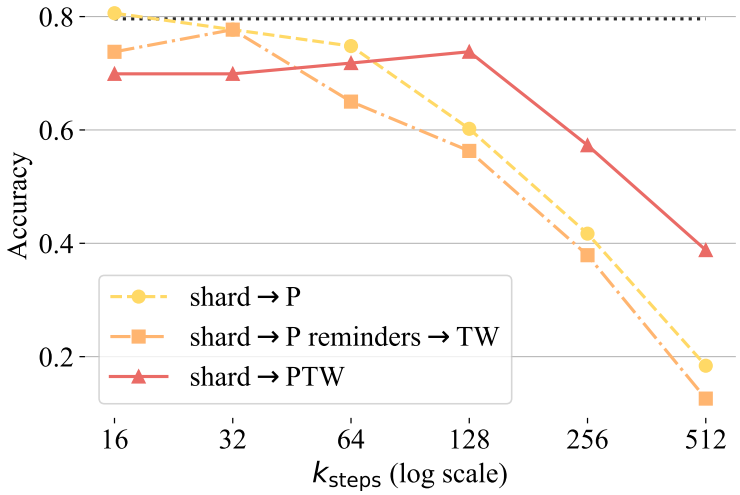

Figure 11: Accuracy with respect to k_{steps} on Qwen3-32B. We report three setups: shard in prompts, shard in all three blocks, shard in prompt and reminder to the thinker and writer on `lost_in_conversations`. The dotted horizontal line denotes the upper bound, where all shards are provided at the start.

P COMPUTE RESOURCES

AsyncReasoning is a training-free inference-time method: no model training or fine-tuning was performed for this work. Consequently, all computational resources were used for inference, evaluation, latency measurements, ablations, and preliminary experiments. Our experiments evaluate existing reasoning LLMs under different inference schedules, including sequential thinking baselines, non-thinking baselines, interleaved variants, and AsyncReasoning variants with concurrent thinking and writing. Since AsyncReasoning does not introduce any gradient computation, optimizer state, or parameter updates, its compute cost is dominated by autoregressive decoding and KV-cache manipulation.

The exact runtime of an individual experiment depends on the model size, benchmark and maximum generation budget. Smaller-scale experiments with Qwen3 models on MATH-500 and MMLU-Pro were typically run on a single 8-GPU server with NVIDIA A100, while larger-model experiments and some latency-focused sweeps were done on other GPUs: H100, H200, B300. All evaluation runs, including Qwen3-235B, fit into 16 CPU cores and 512 GiB system RAM.

The total compute budget varies across experiments (e.g. MMLU-Pro has ≈12K samples, AIME-2025 has 30 samples). Below, we provide several example runtimes for AsyncReasoning evaluations:

1. **Qwen3 235B A22B Thinking 2507 on MATH-500** took 152 B200-hours for AsyncReasoning, 510 B200-hours for baseline-thinking, 139 B200-hours for non-thinking.
2. **Qwen3 32B SpokenMQA multistep-reasoning** took 30-32 A100-hours for either reasoning type.

We conservatively estimate that the main experiments took ≈600K mixed GPU hours, heavily dominated by the runtime of larger models (Qwen3-235B-A22B) in non-asynchronous mode on MMLU-Pro (≈ 12K samples) and k_steps sweeps. This includes pilot experiments, failed runs, implementation debugging, prompt/mode-switching development, and ablations not included in the final paper. The largest share of this budget was spent on repeated inference evaluations for reasoning benchmarks, where each method must be thoroughly evaluated.

The reported numbers should be interpreted as conservative estimates of the research compute used during development, rather than the minimum compute required to reproduce the method. A substantial fraction of early experiments used non-optimized inference code while the efficient concurrent-attention implementation was still under development. As a result, several early runs under-utilized available GPUs due to Python overhead, non-optimal batching, custom attention debugging, and conservative scheduling choices. The optimized implementation included in the supplementary code is expected to require substantially less compute for reproducing the main experiments.

Q LIMITATIONS

Technical limitations. AsyncReasoning is a training-free inference-time method, but it requires explicit control over the model's KV cache, cache-block layout, and positional representations. In the RoPE setting, this includes applying stream-specific query rotations so that the same stored keys and values are interpreted under different Thinker and Writer views. As a result, practical deployment may require custom attention kernels or deeper integration into inference runtimes, rather than relying only on standard sequential decoding APIs.

Mode-switching limitations. AsyncReasoning relies on the model's ability to decide when the Writer can continue and when generation should pause for additional thinking. In our training-free setup, this decision is implemented through prompting, making it dependent on the base model, prompt format, decoding parameters, and task type. Weaker models may switch too early, producing answers before sufficient reasoning has accumulated, or switch too late, reducing latency gains. More robust switching could be achieved with learned routers, verifiers, or task-specific controllers, but these extensions are beyond the purely training-free setting studied here.

Safety limitations. Although AsyncReasoning combined with a safety prompt can let the Thinker interrupt unsafe generations effectively in our experiments, it should not be treated as a complete safety guarantee. Since the Writer may in principle emit public tokens before the Thinker has finished

deliberating, race conditions remain an inherent failure mode of asynchronous safety verification, even if our experiments observe them rarely. Safety performance also remains model- and prompt-dependent. Therefore, AsyncReasoning should be combined with output-side safety filters, refusal policies, and hard stopping mechanisms that can immediately interrupt Writer generation.

R IMPACT STATEMENT

Potential benefits. Real-time systems often trade off either interactivity (fast but shallow responses) or reasoning depth (slow, non-interactive read–think–answer). By allowing models to speak while still reasoning, AsyncReasoning can improve responsiveness in time-sensitive human–AI interaction, reduce the need to interrupt and discard partial reasoning when new information arrives, and make iterative clarification workflows more natural. These properties may be particularly beneficial for assistive technologies (e.g., accessibility-oriented voice interfaces), operational support tools that must react to changing user constraints, and interactive decision-support where users progressively refine goals and constraints.

Safety and alignment considerations. A core risk in real-time generation is that lowering latency can also lower the opportunity for safety deliberation. This work explicitly studies safety-oriented prompting where the model reasons privately about safety implications while streaming benign content, and can pause potentially harmful outputs when additional internal deliberation is needed. In the best case, this can improve the safety–latency trade-off relative to purely non-thinking baselines by enabling more deliberative filtering without fully sacrificing interactivity. However, AsyncReasoning does not guarantee safer behavior: safety performance remains model- and prompt-dependent, and in some cases faster streaming could increase the chance that unsafe or misleading partial outputs are produced before the system detects the need to pause. Deployments should therefore treat AsyncReasoning as a *complement* to standard safety mitigations (policy filters, refusal training, monitoring, and red-teaming), not a replacement.

Additional human inputs: opportunities and risks. Supporting mid-generation inputs can strengthen human oversight by enabling users to correct mistakes, add constraints, or halt undesired trajectories without restarting an interaction. At the same time, streaming inputs can introduce new failure modes: adversarial or accidental late-arriving instructions may steer the model toward policy-violating behavior, and partially generated content may be misinterpreted by users as final or fully-considered. Systems should clearly communicate when outputs are provisional, provide UI affordances for interruption/rollback, and log or surface when late inputs materially change the model's plan.

Misuse and dual-use. Improved real-time capability can be misused in settings such as persuasive social engineering, automated harassment at scale, or interactive assistance for wrongdoing. While the method itself is training-free and model-agnostic, enabling more responsive agentic behavior can increase the attractiveness of deploying powerful models in high-stakes contexts. Practical safeguards (rate limits, content moderation, audit trails, and conservative defaults for safety mode-switching) are important when integrating this technique into user-facing products.

Privacy and data handling. Real-time and mid-stream inputs may include sensitive personal information (spoken or typed). Deployments should minimize retention of raw streams, apply encryption in transit and at rest, and provide clear user controls over logging and deletion. If used in voice assistant pipelines, additional privacy risks arise from continuous capture and transcription, which should be addressed at the system level (on-device processing where feasible, explicit recording indicators, and least-privilege access).

Environmental and efficiency impacts. AsyncReasoning targets *latency* and interactivity rather than reducing total compute in all cases, and may encourage broader use of reasoning-heavy models in real-time applications. Wider adoption could increase overall inference demand; conversely, improved responsiveness may reduce repeated restarts and duplicated computation in interactive sessions.

Overall, we view AsyncReasoning as enabling more natural interactive reasoning systems while introducing new safety, privacy, and misuse considerations that must be addressed with careful prompting, product design, and standard deployment safeguards.